National Library of Canada
Canadian Theses Service
Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Services des thèses canadiennes

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ MICROFILMÉE TELLE QUE NOUS L'AVONS REÇUE

NL-339 (r. 86/06)

Canada

The University of Alberta

PARTITIONING A LOGICAL MODEL OF DATA IN AN ENTERPRISE INTO SUBJECT DATABASES

by

Vladimir Gertsberg

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
Spring, 1987

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-37747-X

THE UNIVERSITY OF ALBERTA

— *RELEASE FORM* —

NAME OF AUTHOR: Vladimir Gertsberg.

TITLE OF THESIS: Partitioning a Logical Model of Data in an Enterprise into Subject Databases.

DEGREE FOR WHICH THIS THESIS WAS PRESENTED: Master of Science

YEAR THIS DEGREE GRANTED: 1987

Permission is hereby granted to The University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(Signed) .....

Permanent Address:
#209 17108-64 Avenue
Edmonton, Alberta
Canada T5T 2C8

Dated 17 February 1987

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled **Partitioning a Logical Model of Data in an Enterprise into Subject Databases.** submitted by **Vladimir Gertsberg** in partial fulfillment of the requirements for the degree of **Master of Science.**

Wm. W. Armstrong

Supervisor

W Bolos

Stan Cabay

Lorna Stewart

Paul Dansey

Date 17.02.1987

ABSTRACT

This thesis analyzes the problem of partitioning a set of entities into disjoint subject databases of limited size so as to minimize various measures of costs for development and maintenance. Formally, the set of entities and the set of transactions define a hypergraph whose nodes are entities and whose edges are transactions, and the whole problem is reduced to partitioning the set of nodes of the hypergraph into subsets of limited sizes so as to minimize the sum of the costs of all cut edges. This thesis analyzes the complexity of different classes of partitioning problems and the properties of some heuristic algorithms, and will present an algorithm for exact solution of a special case of the partitioning problem.

Acknowledgements

I would like to thank my supervisor, William Armstrong, for his guidance and support throughout this research.

Table of Contents

Chapter	Page
Chapter 1: Introduction	1
1.1. A Maintenance Model	3
1.2. The Partitioning Problem and its Applications	7
1.3. Review of Partitioning Methods	10
1.3.1. Exact Methods	10
1.3.2. Random Solutions	11
1.3.3. Clustering Methods	11
1.3.4. Partitioning Methods Based on Minimal Groups	12
1.3.5. λ-Opting Methods	13
1.4. Time Complexity of Algorithms	13
1.5. Decision and Optimization Problems	14
1.6. NP-completeness, NP-hardness and Polynomial Reducibility	15
1.7. Absolute Approximation Algorithms	16
1.8. ϵ-Approximation Algorithms	17
Chapter 2: The General Partitioning Problem	18
2.1. Complexity of the General Partitioning Problem	18
2.2. Absolute Approximation to the General Partitioning Problem	21
2.3. ϵ-Approximation to the General Partitioning Problem	27
Chapter 3: The Bipartite Partitioning Problem	32
3.1. NP-completeness of the Bipartite Partitioning Problem	32
3.2. Absolute and ϵ-Approximations to Bipartite Partitioning	38
Chapter 4: The Tree Partitioning Problem	40
4.1. Basic Definitions and Statement of the Problem	40
4.2. Tree Partitioning With an Unlimited Number of Clusters	42
4.2.1. Statement of the Algorithm	50
4.2.2. Proof of Optimality of the Partition	54
4.2.3. Complexity of the Partitioning Algorithm	64
4.3. Tree Partitioning With a Limited Number of Clusters	70
Chapter 5: Review of Database Design Methods	72
5.1. Database Design Objectives	72
5.2. Inputs to Database Design	74
5.2.1. The Entity-Relationship Diagram	74
5.2.2. Structure of a Transaction	75
5.3. High-Level Logical Design	77
5.4. Physical Database Design	79
Chapter 6: Conclusions	81
References	83

A1: Solution of the Differential Equation 85
A2: Example of Partitioning of the Tree. 87

List of Figures

Figure		Page
1.1	Sets V, R, and minimal group S	12
2.1	Instance of the graph G*	23
2.2	Instance of the graph G constructed from the graph G*	24
2.3	Graph $G(V,E)$	29
3.1	Complete bipartite subgraph S_i	34
3.2	Instance of the graph partitioning problem	35
3.3	Instance of the bipartite partitioning problem	35
4.1	Tree with enumerated nodes	43
4.2	Example of the tree $R(v)$	45
4.3	Transaction trees	46
4.4	Data structure to store partition	65

Chapter 1

Introduction

The latest achievements in hardware performance and software design have led to the development of new Fourth Generation tools which are being successfully used in all stages of design and development of large commercial and engineering computer systems. These new tools, mainly relational database management systems, active data dictionaries and high-level programming languages, make it possible to establish a whole new approach to system design and development. New system development methodologies, known as Fourth Generation Methodologies, have appeared on the market. One of the most successful and most used fourth generation methodologies is the so-called data driven prototyping methodology.

This prototyping methodology takes advantage of these new tools that allow simple and fast data definition and loading. It also provides powerful fourth generation languages for fast program development. Prototyping is an iterative process of system development. Each iteration is used to develop a working prototype of the new system. The complexity of the prototype and the number of implemented system functions grow from stage to stage, so that the first stage usually presents screen design and primitive data flow, while the last prototype presents the production version of the new system.

This approach requires the participation of users during the early stages of the system design, so that they will see from the very beginning what the system will look like and how it will operate. Such participation will minimize the number of changes that must be made during the final stages of system implementation. However, a stable *data model* must be in place from the very beginning of the project, utilizing this new approach. This fact puts new emphasis on the whole data modelling process, because the success of the project becomes dependent on the quality and adequacy of

the data model produced during design.

Data modelling[18,21] is the process of identifying *entities* and *entity relationships* which are relevant to the information system being designed. The concept of entity is very general and can include any object, phenomenon, property or statement relevant to the modelling environment. Entities participate in relationships. For example, one statement about the parent can be associated with several statements about children. The theory of normalization developed by E.F. Codd[2,3] places some restrictions on the type of statements that can be used in the data model. Namely, it requires the normalization of the relations used to represent the statements.

The data model for a contemporary large information system can consist of hundreds of different entities. The information system must be able to process hundreds of different transactions, each accessing a different subset of entities. Practical experience shows that, in these circumstances, any attempt to develop the whole system at once usually ends in failure: the communication overhead goes up, the clean structure of the system disappears, the project becomes unmanageable and so on. Sometimes it is also impossible to put in place all the resources which are necessary for the development of the whole system at once.

The above-mentioned problems show the necessity for partitioning a large system into a set of several manageable subsystems with defined interfaces. The problem of partitioning a large application system into several subsystems is of great importance to any organization which conducts *Strategic Data Planning* study [18]. In practice, the partitioning of the system can be carried out in three steps:

Step 1. Partition the set of entities from the data model into several disjoint subsets of closely related entities. Such subsets are called *subject databases*.

Step 2. For each subject database, identify the set of transactions that access entities only from this subject database. Such sets of transactions are called

internal transactions.

Step 3. For each subject database, identify the set of transactions that access entities from this subject database and entities from any other subject database. These transactions are called *interfacing transactions* or *interfaces*.

We shall identify a *subsystem* as a subject database plus its associated internal transactions and associated interfacing transactions. The problem of partitioning is to identify subject databases in such a way that the complexity of the interfaces is reduced to minimum. This approach is in line with structured system development methodology[22] which requires partitioning the system into highly cohesive and loosely coupled parts. This kind of partition will result in reduction of the total overhead for system development and maintenance and will allow for optimal structuring of the development and maintenance teams. Each team will be responsible for a separate subject database, the associated subsystem and its interfaces. Interfacing transactions will, therefore, be the joint responsibility of two or more different teams. Thus, if we reduce the complexity of the interface, the total overhead in system learning and team communication is reduced at the same time. The next section will discuss these topics in more detail.

1.1. A Maintenance Model

In this section, we shall present a simple mathematical model of the maintenance process for a large application computer system. Using this model, we shall show that partitioning the system into smaller subsystems and minimizing subsystem interfaces will lead to a reduction in overall system maintenance cost. The same conclusion can also be derived for the system development cost.

As we know from experience, large application software packages are never static but continuously undergo changes and enhancements. A request for system maintenance can be the result, for example, of a discovered system bug, a change in user

requirements, or a change in legislation. At present, more than 80% of all computer analysts' time is spent on maintenance.

The performance of each maintenance analyst, that is, the number of unit-sized maintenance requests the analyst can process in a unit of time, depends on the level of knowledge that he/she has about the system. Let S be the size of the subsystem assigned to some maintenance analyst. S can be measured in different units (e.g. lines of code, number of entities, number of attributes, etc.) depending on the system under consideration. The maintenance analyst has a working knowledge of several parts of the subsystem. Let W be the total size of these parts. The time required to process a request for maintenance depends on whether this request is related to the part of the subsystem known to the analyst or not. Let t be the time required to make requested changes to a unit-sized part of the subsystem known to the analyst, and let T be the time per unit-sized part required to change parts unknown to him/her. It is clear that $T >> t$.

We assume that each analyst receives a steady flow of maintenance requests and that each request refers to a specific part of the subsystem with probability proportional to the size of this part. Also, we assume that each analyst receives enough maintenance requests to keep him/her busy all the time. This means that the average time that the analyst spends to process a single unit-sized request is

$$t^* = \frac{W}{S}t + \frac{S-W}{S}T$$

where $\frac{W}{S}$ is the expected portion of the requested task in the parts known to the analyst, and $\frac{S-W}{S}$ is the expected portion of the requested task in the parts unknown to the analyst.

The performance, P, of the analyst in unit-sized tasks per unit time will be equal to

$$P = \frac{1}{t^*} = \frac{S}{Wt + T(S-W)}.$$

The analyst will be forgetting details of the subsystem at a rate proportional to W. At the same time, the analyst will be acquiring knowledge of the subsystem while working with the requests dealing with the unknown parts of the system. The number of unit-sized tasks of the latter type that the analyst processes in a unit of time is equal to $\frac{S-W}{S}P$. This means [1] that the amount of knowledge acquired by the analyst in a unit of time is proportional to $\frac{S-W}{S}P$. These observations allow us to derive a differential equation for W:

$$\dot{W} = -kW + f\frac{S-W}{S}P$$

where k and f are positive coefficients. A time independent solution of this differential equation for the large values of S is derived in Appendix 1. Here, we give only the final expression for the performance of the maintenance analyst:

$$P = \frac{1}{T}(1 + \frac{f(T-t)}{kST^2}).$$

This expression means that if the analyst is assigned to maintain an unreasonably large system (or subsystem), then the analyst's performance will be close to the worst possible performance, $\frac{1}{T}$. Therefore, in order to maintain the performance of the members of the maintenance team above a certain level, we must limit the size of the subsystem to which each analyst is assigned. This, in turn, means that, in order to successfully maintain a large system, we must partition it into several subsystems of limited size.

In order to measure sizes of subsystems and subject databases, we shall assign a size to each entity and each transaction. The size of a subject database will be equal to the total size of all entities in this subject database. The size of a subsystem will be equal to the size of an associated subject database plus the total size of all associated

internal transactions and the total size of all interfacing transactions. However, for the sake of simplicity, we shall assume that the size of a subsystem is proportional to the size of an associated subject database. This is a reasonable assumption when all transactions are evenly distributed throughout the system. This assumption will allow us to reduce the problem of partitioning a system into subsystems of limited size to the problem of partitioning a data model into several subject databases of limited size.

Also, we assume that the amount of system maintenance work is proportional to the system size. However, maintenance of an interfacing transaction will require participation of all teams responsible for subsystems which are accessed by the interfacing transaction, because all such teams must analyze the impact of the requested changes on their subsystems and make the necessary changes. This, however, creates an interface overhead in system maintenance which depends on the sizes of all interfacing transactions and the way these transactions interface with the subsystems. Usually, we shall assume that this interface overhead is proportional to the total size of the interfacing transactions. Clearly, to minimize the maintenance cost, we must find a partition which minimizes the total size of the interfacing transactions.

Suppose that the data model of the system is partitioned into several subject databases of approximately the same size $S_{database}$. This, in turn, will lead to the partition of the system into several subsystems of approximately the same size $S_{subsystem}$. The productivity P of maintenance analysts is a function of $S_{subsystem}$ which is, according to our assumptions, proportional to $S_{database}$. This allows us to consider P to be a function of $S_{database}$. Let S_{system} be the total system size without interface overhead, let $S_{overhead}$ be the interface overhead, and let L be the total size of interfacing transactions. According to our assumptions

$$S_{overhead} = aL,$$

where a is a positive constant. The number of systems analysts required to maintain

the system is equal to:

$$N_{analysts} = b \frac{S_{system} + aL}{P(S_{database})}$$

where b is a positive constant. The last expression shows that in order to decrease $N_{analysts}$ we must decrease the interface size L and increase the productivity P. We can increase the productivity P by decreasing the size $S_{database}$ of the subject databases. However, we can not decrease size $S_{database}$ indefinitely. Small subject databases lead to high system fragmentation and high interface size. An appropriate selection of the subject database size $S_{database}$ is a problem of its own, and we do not address it here. It is clear, however, that once the value of $S_{database}$ is established, the data model must be partitioned into several subject databases with minimal interface size.

The results of this section should be considered to be qualitative rather than quantitative, and the purpose of this section was to reach only two important conclusions:

First: The necessity for partitioning a data model into subject databases of limited size, and

Second: The necessity for minimizing the total size of interfacing transactions.

The formal definition of the partitioning problem is given in the next section.

1.2. The Partitioning Problem and its Applications

The problem of partitioning a set of entities into subject databases can be formulated in the following general way. Assume that we have a finite set N of entities and a finite set T of transactions. Each entity $n \in N$ is assigned a positive integer $w(n) \geq 0$ called an *entity weight*, and each transaction $t \in T$ is assigned a positive integer $l(t) \geq 0$ called a *transaction weight* or a *transaction cost*. Each transaction t has associated with it a set of entities that this transaction accesses. We shall assume that each transaction accesses a different set of entities. However, if this is not the

case, and two transactions, t_1 and t_2, access the same set of entities, then we shall replace these two transactions by a new transaction t, such that

$$l(t) = l(t_1) + l(t_2).$$

This kind of transformation will have no impact on the generality of our analysis, but it will allow us to identify each transaction t with the set of entities that this transaction accesses. We shall refer to the set of entities associated with the transaction t also as t.

A *partition* of N is defined as a collection of subsets of N, called subject databases, $N_1, N_2, \ldots, N_M$ such that

$$N_1 \cup N_2 \cup , \ldots , \cup N_M = N$$
$$N_i \cap N_j = \emptyset \quad \text{for} \quad i \neq j, \quad 1 \leq i,j \leq M.$$

Each subject database N_i will have an associated weight (or size) $W(N_i)$ such that

$$W(N_i) = \sum_{n \in N_i} w(n), \quad i=1,2,\ldots,M.$$

A weight constraint will be imposed on each subject database such that

$$W(N_i) \leq W, \quad i=1,2,\ldots,M$$

where W is a positive integer. The weight constraint is necessary to limit the size of subject databases and associated subsystems to maintain productivity of analysts on a certain level, as discussed in the previous section.

A partition satisfying these weight constraints is called *a feasible partition*. A transaction $t \in T$ is said to be cut by the partition if it accesses entities from different subject databases. We shall define the cost L of the partition as the total cost of all transactions cut by the partition. That is, if $T' \subseteq T$ is the set of all transactions cut by the partition, then

$$L = \sum_{t \in T'} l(t).$$

The value of L represents the interface overhead defined in the previous section.

Now we can formulate the problem of partitioning a set of entities into subject databases as the problem of finding a feasible partition with minimal cost. A feasible partition with minimal cost is called an *optimal partition.*

The problem can be easily formulated in terms of hypergraphs, where nodes of the hypergraph correspond to entities and edges of the hypergraph correspond to transactions.

The partitioning problem has many different applications. One application is placing the components of an electrical circuit onto printed circuit cards with a minimum number of connections between the cards. Here, electrical components correspond to the nodes (our entities) of the hypergraph, circuit connections correspond to the hypergraph edges (our transactions) and cards correspond to the subsets (our subject databases) of the hypergraph nodes. There will be some constraints on the number of electrical components that can be placed on one card, which corresponds to the limited size of the subject databases. Also, it is clear that the cost of connections between the cards is generally much greater than the cost of connections within the card; therefore, the latter can be ignored. This means that the cost of the par ition introduced above can represent the cost of interconnecting the cards with electrical components.

The same approach can be used to improve the performance of the programs operating in an environment with paged memory organization. Here, subroutines correspond to nodes, edges correspond to the transfer of control from one subroutine to another, and pages correspond to sets of nodes of limited size. The optimal assignment of subroutines to pages is such that it minimizes the number of references between subroutines located in different pages.

We have shown some representative examples of important applications where partitioning problems naturally arise. Many other such applications can be found in

operations research, database design, pattern recognition, etc.

1.3. Review of Partitioning Methods

As mentioned, the partitioning problem is a part of many important applications. As a result, researchers have proposed many different methods to solve this problem. In this section, we shall present a brief review of different partitioning methods.

1.3.1. Exact Methods

Presently, several established methods can find an exact optimal solution to the partitioning problem. In 1969, L. Gorinshtein [8] suggested an algorithm for graph partitioning based on a branch-and-bound technique. In 1975, J. Lukes [17] developed a partitioning algorithm based on a dynamic programming technique. Both these algorithms find exact optimal partitioning of graphs, but they have an exponential time complexity function, which makes these algorithms rather useless for practical purposes. Therefore, the generalizations of these algorithms are not used to solve partitioning problems for hypergraphs.

Several successful attempts have found some restricted versions of the partitioning problem which can be solved exactly in polynomial time. In 1971, B. Kernighan [13] found a polynomial time algorithm for finding sequential partitions of graphs. Kernighan assigned a sequence number to each node of the graph and assumed that each cluster of the partition consisted of several nodes with consecutive numbers. For example, a graph with ten nodes numbered from 1 to ten can be partitioned into clusters like { (1,2,3,4,5), (6,7), (8,9,10) } or { (1,2,3), (4,5,6), (7,8,9,10) }. In 1974, J. Lukes [16] found an efficient algorithm for partitioning trees.

We shall show later that any algorithm which finds the exact optimal solution of the general partitioning problem will not have polynomial time complexity, unless P=NP. This means that the only practical approach is to use some heuristic algo-

rithm which can find an approximation to the optimal solution of the partitioning problem.

1.3.2. Random Solutions

The approximation method based on random solutions is very simple. We just generate several random solutions and retain the best of them. This approach, however, was found unsatisfactory for problems of even moderate size. The experiments conducted by B. Kernighan and S. Lin [12] showed that there are generally few optimal or near-optimal solutions which, therefore, appear randomly with very low probabilities.

1.3.3. Clustering Methods

Clustering methods are among the most widely used methods for solving partitioning problems. The fundamental work of S. Johnson [10] describes the hierarchical clustering technique, which enables us to group objects into a hierarchical system of clusters on the basis of similarity among these objects. In 1972, W. McCormick, P. Sweitzer and T. White [19] developed the so-called *bond energy algorithm* which allows us to cluster data according to their similarity into several overlapping clusters. The algorithm tends to minimize the size of cluster overlapping. An extensive review of clustering methods can be found in [5, 23].

The main disadvantage of clustering methods is that, in general, they do not include provisions for satisfying constraints on the sizes of resulting clusters. This fact significantly reduces the importance of clustering methods for data model partitioning.

1.3.4. Partitioning Methods Based on Minimal Groups

An interesting method for data partitioning was suggested by F. Luccio and M. Sami in 1969 [15] and further extended by J. Kaoprzyk and W. Stanozak in 1976 [11]. The method is based on so-called *minimal groups*. Let $f(A,B)$ be a non-negative measure of similarity between groups of objects A and B. We define a minimal group as follows:

Definition: Let V be the set of objects and let $S \subseteq V$ be a non-empty group of objects. If the following inequality

$$f(R,S-R) > f(R,V-S)$$

holds for every non-empty subgroup R of a group S, then S is called a *minimal group*.

The interpretation of this inequality is very natural. It assumes that inner similarity, i.e. strength of similarity between sets R and $S-R$, is stronger than similarity between a part of a group and the rest of the objects from outside a considered group, i.e. strength of similarity between sets R and $V-S$ (see also Figure 1.1).

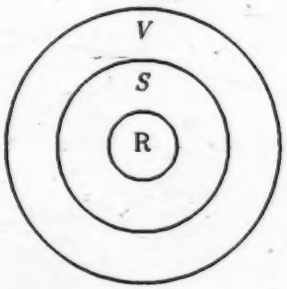

Figure 1.1 Sets V, R, and minimal group S

The algorithm based on the concept of minimal groups has not received much recognition from computer practitioners, primarily because the algorithm has exponential time complexity and there is no way to control the sizes of minimal groups.

1.3.5. λ-Opting Methods

S. Lin, working on the Traveling Salesman Problem [14], suggested a set of methods for improving a given solution by rearranging single links, double links and in general λ links. The configuration of the system is called λ-opt (i.e. optimal) if no exchange of λ links results in a configuration with lower cost.

The same idea can be used in partitioning a set of objects into smaller subsets. We can rearrange a given partition by moving 1, 2 or up to λ objects from one subset to another, and we can also define λ-opt partitions as those which are optimal with respect to rearrangements involving movement of up to λ objects.

Unfortunately, experiments conducted by S. Lin and B. Kernighan [12] showed the poor performance of λ-opt algorithms for solving partitioning problems.

This section concludes the review of partitioning methods. We shall devote the rest of the chapter to introducing some concepts and definitions which will be used for analysis and design of partitioning algorithms. These definitions are taken primarily from [6,9].

1.4. Time Complexity of Algorithms

The *time complexity* of an algorithm expresses its execution requirements by giving, for each possible input length, the largest amount of time needed to solve any problem instance of that size. It is clear that time requirements will depend on the encoding scheme used to determine input length and the computer model used to solve the problem. However, as long as resulting time complexity functions vary by no more than a polynomial function, this fact will have no impact on our analysis.

In the rest of the thesis, we shall use the following concepts in analyzing the time complexity of algorithms:

Definition: $f(n) = O(g(n))$ iff there exist two positive constants c and n_0 such that

$|f(n)| \leq c|g(n)|$ for all $n \geq n_0$.

Definition: An algorithm Ω with the time complexity function $f(n)$ is called *a polynomial time algorithm* if the time complexity $f(n) = O(p(n))$ where n is the input length of the problem and $p(n)$ is a polynomial function. Polynomial time algorithms are sometimes called *fast* or *efficient*.

Definition: An algorithm Ω with the time complexity function $f(n)$ is called *an exponential time algorithm* iff there exist positive constant $c>1$ such that $f(n) \geq c^n$ for an infinite number of values of n.

1.5. Decision and Optimization Problems

Our analysis will be applied to two different versions of the partitioning problem, namely the *decision problem* and the *optimization problem*.

Definition: The following problem is called the hypergraph partitioning optimization problem.

Instance: Hypergraph $H(V,E)$, weights $w(v) \geq 0$ for all $v \in V$, $l(e) \geq 0$ for all $e \in E$, and positive integer W.

Problem: Find a partition $V_1, V_2, \ldots, V_M$ of the set V with a minimal cost $L = \sum_{e \in E'} l(e)$, where E' is the set of edges not contained in some V_j, and such that

$$\sum_{v \in V_i} w(v) \leq W \quad \text{for} \quad 1 \leq i \leq M.$$

Definition: The following problem is called the hypergraph partitioning decision problem.

Instance: Hypergraph $H(V,E)$, weights $w(v) \geq 0$ for all $v \in V$, $l(e) \geq 0$ for all $e \in E$, and positive integers W and L.

Question: Is there a feasible partition of V with the cost less than or equal to L and such that

$$\sum_{v \in V_i} w(v) \leq W \quad \text{for} \quad 1 \leq i \leq M?$$

The optimization problems are at least as hard as the corresponding decision problems, because once the optimization problem is solved, the answer to the decision problem can be found immediately by comparing the optimal cost with the given bound L. The reverse statement is not always true, although some optimization problems have the same time complexity as the corresponding decision problems.

1.6. NP-completeness, NP-hardness and Polynomial Reducibility

This section will contain basic definitions used in the classification of the algorithms.

We will define P as the class of problems solvable by deterministic algorithms in polynomial time. NP is the class of all problems solvable by nondeterministic algorithms in polynomial time [6]. Deterministic algorithms are a special case of nondeterministic algorithms, therefore, $P \subseteq NP$. One of the most famous unresolved problems of computing science is whether $P = NP$ or $P \neq NP$.

Each decision problem has associated with it a set of problem instances which result in a "Yes" decision. This is the so-called set of *yes-instances* of the decision problem. Now we can introduce the concept of polynomial reducibility.

Definition: A decision problem L_1 is *reducible* to a decision problem L_2 if there exists a constructive transformation which converts any instance I_1 of L_1 into a corresponding instance I_2 of L_2 such that I_1 is a yes-instance of L_1 if and only if I_2 is a yes-instance of L_2. If the transformation can be constructed in time bounded by a polynomial function of the input size, then L_1 is *polynomially reducible* to L_2.

In this thesis we shall be interested only in a polynomial reducibility, so there will be no confusion if we shall use the term "reducible" instead of "polynomially reducible". This definition of reducibility implies that if we have a polynomial time algo-

rithm for the problem L_2, then we can also solve the problem L_1 in polynomial time.

In 1971, Cook [4] proved that every decision problem in *NP* can be polynomially reduced to one special problem in *NP* called ***satisfiability***. Subsequently, it was found that many other decision problems share this property with the satisfiability problem. The class of such problems was given a special name, *NP−complete*. The following definition formalizes this concept:

Definition: A problem is *NP−complete* if it is in *NP* and every problem in *NP* can be polynomially reduced to it.

Due to the transitivity of polynomial reduction, to show that some new problem Π is *NP−complete* it is sufficient to prove that

1) Π is in *NP* and

2) There is an NP-complete problem that is polynomially reducible to Π.

The concept of NP-completeness will be used mainly to analyze decision problems. To analyze optimization problems, we shall use the concept of *NP−hardness*.

Definition: A problem Π is *NP−hard* if every problem in NP can be polynomially reduced to it.

We do not require NP-hard problems to belong to NP. It is easy to see that all NP-complete problems are NP-hard, but there are NP-hard problems that are not NP-complete. The *halting* problem, for example, is NP-hard but not NP-complete.

1.7. Absolute Approximation Algorithms

If some optimization problem is found to be NP-hard, then, unless $P=NP$, this problem does not have a polynomial time algorithm. In practice, such problems will usually be solved only approximately by applying some heuristic algorithms. In this section, we shall define the term ***absolute approximation algorithm***.

Definition. Let Ω be an algorithm which generates a feasible solution to every instance

I of problem Π. Let $F^*(I)$ be the value of an optimal solution to I and let $F(I)$ be the value of a feasible solution to I generated by Ω. The algorithm Ω is called an *absolute approximation algorithm* for the problem Π iff for every instance I of Π, $|F^*(I) - F(I)| \leq k$ for some constant k.

An absolute approximation algorithm is one of the most desired types of heuristics. However, not all optimization problems have polynomial time absolute approximation algorithms.

1.8. ϵ-Approximation Algorithms

An ϵ-approximation algorithm is another type of heuristic which is sometimes easier to find than an absolute approximation algorithm. This type of heuristic is formally defined as follows:

Definition. Let Ω be an algorithm which generates a feasible solution to every instance I of problem Π. Let $F^*(I)$ be the value of the optimal solution to I and let $F(I)$ be the value of the feasible solution to I generated by Ω. The algorithm Ω is called an *ϵ-approximation algorithm* for problem Π iff for every instance I of Π,

$$\frac{|F(I) - F^*(I)|}{F^*(I)} \leq \epsilon.$$

Chapter 2

The General Partitioning Problem

This chapter is devoted to the analysis of the time complexity of the decision partitioning problem and to the discussion of the properties of absolute and ϵ-approximation heuristics to the optimization partitioning problem.

2.1. Complexity of the General Partitioning Problem

In this section we shall discuss different versions of the entity set partitioning problem and shall prove that their decision problems are NP-complete.

Theorem 1. The following decision problem is NP-complete.

Instance: Set of entities N, set of transactions T, weights $w(n) \geq 0$ for each $n \in N$, costs $l(t) \geq 0$ for each $t \in T$, and positive integers W, L.

Question: Is there a partition of N into disjoint sets $N_1, N_2, \ldots, N_M$ such that $\sum_{n \in N_i} w(n) \leq W$ for $1 \leq i \leq M$ and such that if $T' \subseteq T$ is the set of all transactions in T that access entities from at least two different sets N_i, then $\sum_{t \in T'} l(t) \leq L$?

Proof. It is easy to see that the problem is in NP because it is possible to verify in polynomial time whether any partition satisfies the theorem's conditions or not. In order to show that the problem is NP-complete, we shall reduce the graph partitioning problem to it.

The graph partitioning problem is formulated as follows [6]:

Instance: An undirected graph $G(V,E)$ and positive integers K and J.

Question: Is there a partition of V into disjoint sets $V_1, V_2, \ldots, V_M$ such that $|V_i| \leq K$ for $1 \leq i \leq M$ and such that if $E' \subseteq E$ is the set of edges that have their two endpoints in different sets of the partition, then $|E'| \leq J$?

Let us take any instance of the graph partitioning problem. For each vertex in V we shall create one entity in N, thus establishing a one-to-one correspondence between

vertices in V and entities in N. For each edge in E connecting two vertices, we shall create a transaction in T accessing the corresponding entities, and thus establish a one-to-one correspondence between edges in E and transactions in T.

We also assume that

$$w(n) = 1 \quad \text{for} \quad n \in N,$$
$$l(t) = 1 \quad \text{for} \quad t \in T,$$
$$W = K, \quad \text{and}$$
$$L = J.$$

Now it is obvious that if there is a partition of V satisfying the conditions of the graph partitioning problem, then the corresponding partition of the set of entities N will satisfy the conditions of Theorem 1.

The opposite statement is also valid. If there is a partition of the set of entities N satisfying the theorem's conditions, then the corresponding partition of V will satisfy the conditions of the graph partitioning problem.

The time complexity of the transformation employed is polynomial, which means that the graph partitioning problem is polynomially reducible to the entity set partitioning problem. This concludes the proof of Theorem 1.

From the way the theorem has been proven, it is clear that the entity set partitioning problem will remain NP-complete even if each transaction accesses no more than two entities and the weight of each entity and the cost of each transaction are equal to one.

Slightly different formulations of the entity set partitioning problem must often be considered. For example, we can limit the number of subject databases. The new partitioning problem will be formulated as follows:

Instance: Set of entities N, set of transactions T, weights $w(n) \geq 0$ for each $n \in N$, and costs $l(t) \geq 0$ for each $t \in T$, positive integers W, L, M.

Question: Is there a partition of N into no more than M disjoint sets $N_1, N_2, \ldots, N_M$

such that $\sum_{n \in N_i} w(n) \leq W$ for $1 \leq i \leq M$ and such that if $T' \subseteq T$ is the set of all transactions that access entities from the different sets of the partition, then $\sum_{t \in T'} l(t) \leq L$?

This new problem has some additional restrictions on the solution but not on the input data. In general, additional restrictions on the output data make the problem "harder" in a sense that if we have an algorithm for solving this problem, then the same algorithm can be used to solve the original problem without any additional transformations. This applies to our problem as well. Suppose that we have an algorithm to solve the entity set partitioning decision problem where M is the maximal number of subject databases allowed in the partition. If we set $M = |V|$, which effectively removes any limitations on the number of subject databases, then we shall reduce our original problem to the new one. This means that the decision problem for entity set partitioning with a limited number of subject databases is also NP-complete.

We can consider another interesting formulation of the entity set partitioning problem with a differently calculated cost of partitioning. The new cost L of the partitioning will be calculated as

$$L = \sum_{t \in T} f(t) \times l(t),$$

where $f(t)$ is the number of the subject databases that transaction t accesses.

Formally, this version of the entity set partitioning problem can be formulated in the following way:

Instance: Set of entities N, set of transactions T, weights $w(n) \geq 0$ for each $n \in N$, and costs $l(t) \geq 0$ for each $t \in T$, positive integers W, L.

Question: Is there a partition of N into disjoint sets $N_1, N_2, \ldots, N_M$ such that $\sum_{n \in N_i} w(n) \leq W$ for $1 \leq i \leq M$ and such that $\sum_{t \in T} f(t) \times l(t) \leq L$, where $f(t)$ is the number of the subject databases that transaction t accesses?

It is easy to see that the problem is NP-complete because graph partitioning can be reduced to it in a very straightforward manner. As in Theorem 1, for each node v_i in V we shall create entity n_i in N, and for each edge $e_{ij} = (v_i, v_j)$ in E we shall create transaction $t \in T$ accessing corresponding entities n_i and n_j. We also set

$$w(n) = 1 \quad \text{for} \quad n \in N,$$
$$l(t) = 1 \quad \text{for} \quad t \in T,$$
$$W = K,$$
$$L = J + |T|.$$

The value of $|T|$ is equal to the number of edges in the graph which is constant for the given instance of the graph partitioning problem. The suggested transformation can be done in polynomial time, and it is easy to see that if the answer to the graph partitioning problem is "Yes", then the answer to the entity set partitioning problem is "Yes". The opposite statement is also true. If the answer to the entity set partitioning problem is "Yes", then the answer to the graph partitioning problem is also "Yes". Therefore, this version of the entity set partitioning problem is also NP-complete. Such a partitioning problem can arise in management information system development if we calculate the total cost of transaction development as some basic transaction cost (weight) multiplied by the number of subject databases that this transaction accesses.

2.2. Absolute Approximation to the General Partitioning Problem

In this section we shall discuss the properties of an absolute approximation algorithm for the solution of the optimization partitioning problem. We shall prove that the problem of finding an absolute approximation is NP-hard, even if the weights of all entities and the costs of all transactions are equal to one and each transaction accesses no more than two entities.

Formally, the problem can be stated in the terms of graphs in the following way:

Theorem 2. A polynomial time absolute approximation algorithm to the following

optimization problem can exist only if $P = NP$.

Instance. Graph $G(V,E)$ and positive integer W.

Optimization. Find a partition $V_1, V_2, \ldots, V_m$ of V which minimizes the cardinality of the set Q consisting of all edges having their endpoints in two different sets of the partition and such that $|V_i| \leq W$ for $1 \leq i \leq m$.

Before proceeding with the proof of the theorem, we shall prove the following lemma:

Lemma 1. Let us have a complete graph $G(V,E)$ such that $|V| = J+1$. For any non-trivial partition $V_1, V_2, \ldots, V_m$ of V, the cardinality of the set $Q \subseteq E$ of edges having their endpoints in two different sets of the partition is not less than J.

Proof. Let us take any non-trivial partition $V_1, V_2, \ldots, V_m$ of V. For the non-trivial partition we assume without a loss of generality that

$$1 \leq |V_1| < |V| = J + 1.$$

Let us concatenate subsets V_2 through V_m into subset $\overline{V^*_2}$. It is easy to see that the cardinality of the set $Q^* \subseteq E$ of edges having their endpoints in two different subsets V_1 and V^*_2 is no greater then the cardinality of the set Q, that is

$$|Q^*| \leq |Q|.$$

Now let

$$|V_1| = X$$

then

$$|V^*_2| = J + 1 - X, \text{ and}$$
$$|Q^*| = X \times (J + 1 - X).$$

The expression for $|Q^*|$ is a quadratic polynomial. Taking into account that X is an integer and $1 \leq X < J+1$, we can easily see that $|Q^*|$ achieves minimum when $X = 1$ or $X = J$. This means that

$$|Q^*| \geq J,$$

and also

$$|Q| \geq J.$$

This completes the proof of the lemma. Now we are ready to prove Theorem 2. We shall prove that the absolute approximation algorithm for the optimization partitioning problem can be used to solve the optimization partitioning problem exactly.

Suppose that we have a polynomial time absolute approximation algorithm Ω for the partitioning problem. This means that for any graph this algorithm will find the partition such that

$$S - S_{opt} \leq K,$$

where S is the number of edges having their endpoints in two different sets of the found partition, S_{opt} is the number of edges having their endpoints in two different sets of the optimal partition and K is a positive integer.

Now let us take any instance of the graph partitioning problem that is graph $G^*(V^*,E^*)$ and a positive integer bound W*. We shall construct a new instance of the graph partitioning problem that is graph $G(V,E)$ and a positive integer bound W by the following rule. Each node $v^*_i \in V^*$ will be transformed into a complete subgraph $G^c_i(V^c_i,E^c_i)$ in G. Each subgraph $G^c_i(V^c_i,E^c_i)$ will contain $|E^*|\times(K+1)+K+2$ vertices. Each edge $e^*_{ij} = (v^*_i,v^*_j) \in E^*$ will be transformed into $K+1$ edges connecting $K+1$ different vertices in G_i with $K+1$ different vertices in G_j. For example, the graph shown in Figure 2.1

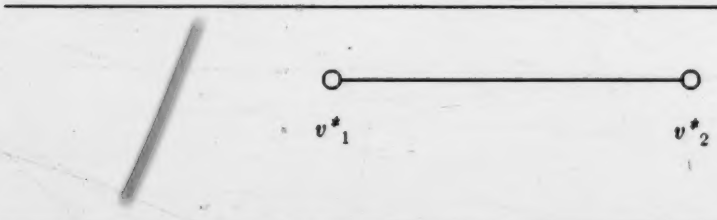

Figure 2.1 Instance of the graph G*

will be transformed into the graph shown in Figure 2.2 for the case when $K=1$.

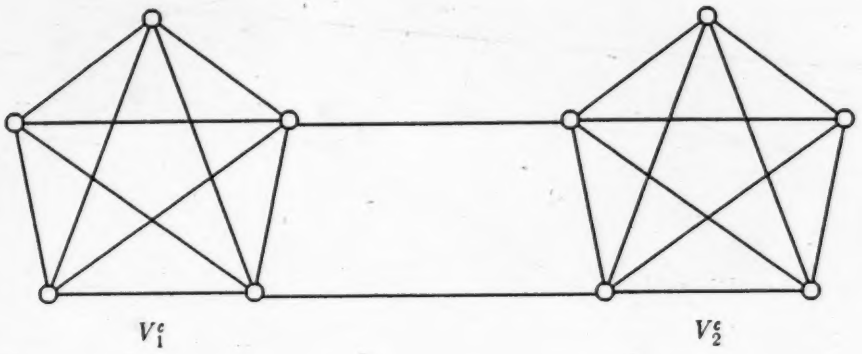

Figure 2.2 Instance of the graph G constructed from the graph G*

We shall also set

$$W = W^* \times (|E^*| \times (K + 1) + K + 2).$$

Let $V_1, V_2, \ldots, V_m$ be the partition of V returned by the algorithm Ω, and let $Q \subseteq E$ be the set of edges having their endpoints in two different sets of the partition. First, let us observe that if we take the partition such that

$$V_i = V_i^c \quad \text{for} \quad 1 \leq i \leq |V^*|$$

then

$$|V_i| \leq W \quad \text{for} \quad 1 \leq i \leq m,$$

and also

$$|Q| = |E^*| \times (K + 1),$$

because all edges in the graph $G(V,E)$ that go from one complete subgraph to another will be cut by the partition, while all edges internal to the complete subgraphs will not be cut.

This means that the value of S which is returned by the algorithm Ω can not be greater than $|E^*|\times(K+1)+K$, otherwise it will differ from the optimal solution by more than K. This, in turn, means that the algorithm Ω cannot split any of the sets V_i^c and each set V_i^c must be contained entirely in one of the sets V_j, otherwise according to the Lemma 1 we will have

$$S \geq |E^*|\times(K+1)+K+1.$$

Obviously, the same conclusion is true for an optimal partition of the set V: an optimal partition cannot split any of the sets V_i^c.

The fact that the algorithm Ω cannot split any of the sets V_i^c means that all edges in $G(V,E)$ which correspond to the edge $e^*_{ij} = (v^*_i, v^*_j) \in E^*$ will belong to the same set V_k or will have one endpoint located in the set V_k, such that $V_i^c \subseteq V_k$, and another endpoint located in the set V_l, such that $V_j^c \subseteq V_l$.

From the above, it follows that if algorithm Ω is applied to the graph $G(V,E)$, then the value of $S=|Q|$ can be represented in the form

$$S = (K+1)\times Z,$$

where Z is a positive integer.

Now we shall prove that

$$S_{opt} = S^*_{opt}\times(K+1).$$

Suppose that we have an optimal partition $V^*_1, V^*_2, \ldots, V^*_m$ of the set V^* with the cost S^*_{opt} and such that

$$|V^*_i| \leq W^* \quad \text{for} \quad 1 \leq i \leq m.$$

Let us create partition $V_1, V_2, \ldots, V_m$ of the set V such that iff

$$v^*_k \in V^*_i$$

then

$$V_k^i \subseteq V_i.$$

It is clear that the cost S of this partition is

$$S = S^*_{opt} \times (K + 1),$$

the cardinalities of the sets V_i satisfy the following inequalities:

$$|V_i| \leq W \quad \text{for} \quad 1 \leq i \leq m,$$

and also

$$S_{opt} \leq S = S^*_{opt} \times (K + 1).$$

On the other hand, let us have an optimal partition $V_1, V_2, \ldots, V_m$ of the set V with the cost S_{opt} and such that

$$|V_i| \leq W \quad \text{for} \quad 1 \leq i \leq m.$$

Using the fact that each of the subsets V_k^i is contained entirely in one of the subsets V_i, we can construct partition $V^*_1, V^*_2, \ldots, V^*_m$ of the set V* such that

$$v^*_k \in V^*_i$$

iff

$$V_k^i \subseteq V_i.$$

It is clear that for this partition

$$|V^*_i| \leq W^* \quad 1 \leq i \leq m, \quad \text{and}$$

$$S^*_{opt} \leq S^* = |Q^*| = \frac{S_{opt}}{(K + 1)}.$$

The last inequality shows that

$$S_{opt} \geq S^*_{opt} \times (K + 1).$$

We have already proven that

$$S_{opt} \leq S^*_{opt} \times (K + 1).$$

The last two inequalities mean that

$$S_{opt} = S^*_{opt} \times (K + 1).$$

We have already proven that the algorithm Ω returns value of S such that

$$S = Z \times (K + 1)$$

and by definition

$$S - S_{opt} \leq K.$$

The last three statements lead to the following statement:

$$Z \times (K + 1) - S^*_{opt} \times (K + 1) = (Z - S^*_{opt}) \times (K + 1) \leq K.$$

This inequality can hold only if

$$S^*_{opt} = Z.$$

This means that the algorithm Ω returns partition of the set V such that

$$S = S^*_{opt} \times (K + 1)$$

and that this partition can be used to construct an optimal partition of the original graph $G^*(V^*,E^*)$. It is evident that all suggested transformations can be done in polynomial time. If algorithm Ω works in polynomial time, then we also can find the optimal partition of the graph in polynomial time. However, this means that P=NP or that the problem of finding an absolute approximation to the general partitioning problem is NP-hard. This concludes the proof of Theorem 2.

2.3. ϵ-Approximation to the General Partitioning Problem

This section will discuss properties of ϵ-approximation algorithms to solve the optimization partitioning problem. We shall prove that finding an ϵ-approximate solution to the general partitioning problem with $0<\epsilon<1$ and with a limitation on the number of subject databases is NP-hard, even if the costs of all transactions are equal to one and each transaction accesses no more then two entities.

Formally, the problem can be stated in the terms of graphs in the following way:

Theorem 3. A polynomial time ϵ-approximation algorithm for $0<\epsilon<1$ to the following optimization problem can exist only if P=NP.

Instance. A connected undirected graph $G(V,E)$, weights $w(v)>0$ for each $v \in V$ and

positive integers W, M.

Optimization. Find a partition of V into no more than M subsets $V_1, V_2, \ldots, V_M$ which minimizes the cardinality of the set Q consisting of all edges having their end-points in two different subsets V_i of the partition and such that $\sum_{v \in V_i} w(v) \leq W$ for $1 \leq i \leq M$.

We shall prove that if an ϵ-approximation algorithm Ω for the graph partitioning problem exists, then it can be used to solve an integer set partitioning problem. The integer set partitioning problem can be formulated as follows [6]:

Instance: A finite set A of positive integers $a_1, a_2, \ldots, a_N$.

Question: Is there a partition P such that

$$\sum_{i \in P} a_i = \sum_{i \notin P} a_i \; ?$$

Let us take any instance of the integer set partitioning problem, that is, a set A of integers. We shall assume that

$$\sum_{i=1}^{N} a_i = 2 \times Y$$

where Y is an integer. If $\sum_{i=1}^{N} a_i$ can not be represented in this way, then we know that the required partition does not exist.

Let us suppose that an ϵ-approximation algorithm for the graph partitioning optimization problem exists and that it always returns partitions such that

$$\frac{S - S_{opt}}{S_{opt}} \leq \epsilon$$

where $S = |Q|$ is the cardinality of the set of the edges cut by the partition, and ϵ is a positive integer less than 1.

For a given instance of the integer set partitioning problem we shall construct the corresponding instance of the graph partitioning problem.

The graph $G(V,E)$ will be constructed in the following way. For each integer $a_i \in A$ we shall create a node v_i such that

$$w(v_i) = a_i \quad \text{for} \quad 1 \leq i \leq N.$$

Also, we shall create two nodes u_1 and u_2 such that

$$w(u_1) = w(u_2) = Y = \tfrac{1}{2}\sum_{i=1}^{N} a_i.$$

Let us take nodes u_1 and u_2 and connect them to each node v_i for $1 \leq i \leq N$. As a result of this construction, we shall obtain complete bipartite graph $G(V,E)$. Figure 2.3 shows such a graph constructed for the set of integers

$$a_1 = 2,\ a_2 = 3,\ a_3 = 4,\ a_4 = 5.$$

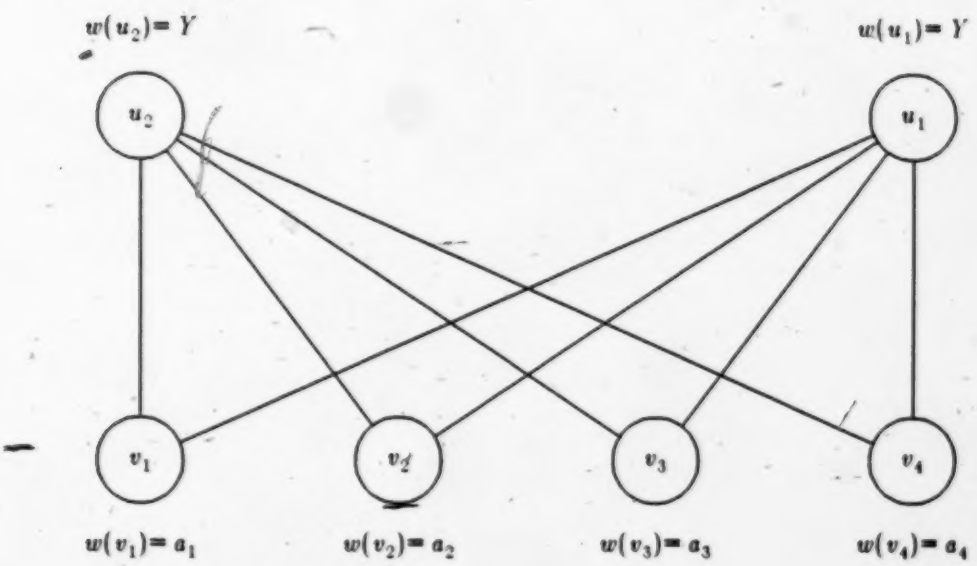

Figure 2.3 Graph $G(V,E)$

To complete the construction of the instance of the graph partitioning problem, we shall also set a bound M on the number of subsets in the partition and bound W on

the total weight of the subsets such that

$$M = 2 \quad \text{and}$$
$$W = 2 \times Y.$$

Suppose that the set of integers A has no partition. Then the only partition of the graph $G(V,E)$ which satisfies the above mentioned constraints is

$$V_1 = v_1, v_2, \ldots, v_N$$
$$V_2 = u_1, u_2$$

with the cost $S = 2N$. This means that the algorithm Ω when applied to the constructed instance of the graph partitioning problem will create a partition with the cost $S = 2N$.

Now suppose that the set of integers A has a partition, and let A_1 and A_2 be such sets that

$$A_1 \bigcap A_2 = \varnothing$$
$$A_1 \bigcup A_2 = A \quad \text{and}$$
$$\sum_{a_i \in A_1} a_i = \sum_{a_i \in A_2} a_i = Y.$$

Now, we can create partition V_1, V_2 of the set V such that set V_1 will contain node u_1 and all nodes v_i which have corresponding integers in the set A_1, while set V_2 will contain node u_2 and all nodes v_i which have corresponding integers in the set A_2. It is clear that the cost of this partition will be equal to N. So, for this case we have

$$S_{opt} \leq N.$$

If we apply the algorithm Ω to the constructed instance of the graph partitioning problem, then this algorithm will return the value of S such that

$$\frac{S - S_{opt}}{S_{opt}} \leq \epsilon,$$

which means that

$$S \leq S_{opt} \times (1 + \epsilon) \leq N \times (1 + \epsilon) < 2 \times N$$

because $0 < \epsilon < 1$.

Now we can see that if the algorithm Ω creates a partition with the cost $S < 2 \times N$, then the set of integers A has a partition. Otherwise, the set A has no partition. The suggested transformation can be done in polynomial time, which means that if the algorithm Ω has a polynomial time complexity, then we can solve the integer set partition problem in a polynomial time. This means, however, that P=NP. This concludes the proof of Theorem 3.

The Theorem 3 is a little bit different from the theorem proved by S. Sahni and T. Gonzalez in 1976 [7]. S. Sahni and T. Gonzalez assigned different weights to each node and each edge to prove NP-hardness of ϵ-approximation. We have proved that the problem will remain NP-hard even if weights of all edges are equal to one.

Chapter 3

The Bipartite Partitioning Problem

This chapter is devoted to the analysis of the bipartite partitioning problem. We shall prove that the general partitioning problem remains NP-complete even if the set of entities consists of two disjoint subsets and each transaction accesses only two entities, each from a different subset. This situation can arise when one of the subsets consists of parent type records, another subset consists of member type records, and each transaction accesses one parent type record and one member type record. Some of the database management systems, for example TOTAL, have this kind of restriction.

We shall also show that a polynomial time absolute approximation algorithm and an ϵ-approximation algorithm to the bipartite partitioning problem can exist only if P=NP.

3.1. NP-completeness of the Bipartite Partitioning Problem

In this section we shall prove that the bipartite partitioning problem is NP-complete, even if the weights of all entities and the costs of all transactions are equal to one. This problem can be formulated in terms of graphs as follows:

Theorem 4. The following decision problem is NP-complete.

Instance: Bipartite graph $G^B(V^B,E^B)$, positive integers W^B,J^B.

Question: Is there a partition of V^B into disjoint sets $V_1^B, V_2^B, \ldots, V_M^B$ such that $|V_i^B| \leq W^B$ for $1 \leq i \leq M$ and such that if $Q^B \subseteq E^B$ is the set of all edges that have their two endpoints in two different sets of the partition, then $|Q^B| \leq J^B$?

First, let us prove the following lemma:

Lemma 2. Let us take a complete bipartite graph $G(V,E)$ such that

$$V^*_1 \cup V^*_2 = V,$$
$$V^*_1 \cap V^*_2 = \emptyset,$$

$$2\times|V^*_1| = 2\times|V^*_2| = |V| = 2\times J,$$

and each edge $e\in E$ has one endpoint in V^*_1 and another endpoint in V^*_2, and all vertices in V^*_1 are connected to all vertices in V^*_2. Then for any non-trivial partition of the set V into disjoint sets $V_1, V_2, \ldots, V_M$, the cardinality of the set of edges Q having their two endpoints in two different sets of the partition is not less than J.

Proof. Let us take any non-trivial partition $V_1, V_2, \ldots, V_M$ of the set V such that $|Q| = S$. We assume that $M\geq 2$ and, also without loss of generality, that

$$1 \leq |V_1| < 2\times J.$$

If we merge any number of subsets V_i, then in the new partition the number of edges having their endpoints in two different subsets of the partition will not be greater than S. Let us merge all subsets V_2 through V_M into a new subset V'_2 and let the number of edges having their endpoints in two different subsets V_1 and V'_2 be equal to S'. As mentioned above, $S'\leq S$.

Now let us introduce some new variables:

$$l_1 = |V_1 \bigcap V^*_1|,$$
$$l_2 = |V'_2 \bigcap V^*_1|,$$
$$n_1 = |V_1 \bigcap V^*_2|, \text{ and}$$
$$n_2 = |V'_2 \bigcap V^*_2|.$$

Using these variables, we can calculate the cost S' of the partition as follows:

$$S' = l_1\times n_2 + l_2\times n_1.$$

From the conditions of the lemma we can conclude that

$$l_1 + l_2 = J,$$
$$n_1 + n_2 = J, \text{ and}$$
$$1\leq l_1 + n_1 < 2\times J.$$

The last inequality means that at least one of the numbers l_1 or n_1 is greater than 0. We assume that $l_1 \geq 1$. In order to prove the lemma, we should prove that

$$S' = l_1 \times n_2 + l_2 \times n_1 \geq J.$$

If $l_2 = 0$, then from the lemma's conditions it follows that $l_1 = J$, $n_1 < J$ and $n_2 \geq 1$. In this case

$$S' = l_1 n_2 + l_2 n_1 = l_1 \times J \geq J.$$

If $l_2 \geq 1$ then

$$S' = l_1 \times n_2 + l_2 \times n_1 \geq n_1 + n_2 = J.$$

So, finally,

$$S \geq S' \geq J.$$

This concludes the proof of the lemma.

To prove Theorem 4, we shall show that the graph partitioning problem can be reduced to the bipartite partitioning problem.

Let us take any instance of the graph partitioning problem. We have graph $G(V,E)$ and positive integers W and J. We should answer the question if there is a partition of the set V into disjoint sets $V_1, V_2, \ldots, V_M$ such that $|V_i| \leq W$ for $1 \leq i \leq M$ and such that, if $Q \subseteq E$ is the set of edges that have their two endpoints in two different sets of the partition, then $|Q| \leq J$.

Having this instance of the graph partitioning problem, we shall construct the instance of the bipartite partitioning problem. The bipartite graph $G^B(V^B,E^B)$ will be constructed in the following way:

For every vertex v_i in V we shall construct a complete bipartite subgraph S_i with $2 \times (J + 1)$ vertices. Such a subgraph for $J=2$ is shown in Figure 3.1. This Figure also explains the meaning of the indices used to name vertices of the subgraph.

For each edge $e = (v_i,v_k) \in E$, we shall create an edge $e^B = (v^B_{1,i,1}, v^B_{2,k,1}) \in E^B$. For example, the graph shown in Figure 3.2 will be transformed into the graph shown in Figure 3.3.

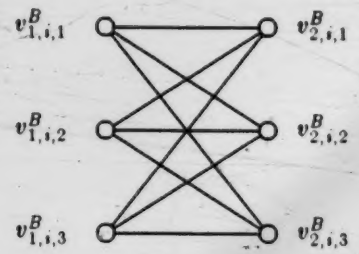

Figure 3.1 Complete bipartite subgraph S_i

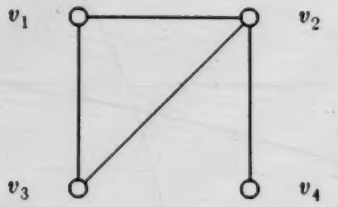

Figure 3.2 Instance of the graph partitioning problem

To complete the transformation of the instance of the graph partitioning problem into the instance of the bipartite partitioning problem we shall also set

$$W^B = 2\times(J + 1)\times W,$$
$$J^B = J.$$

It is obvious that this transformation can be done in polynomial time. Now we shall prove that if the answer to the graph partitioning decision problem is "Yes", then the answer to the bipartite partitioning decision problem is also "Yes".

Let subsets $V_1, V_2, \ldots, V_M$ satisfy the conditions of the graph partitioning problem. Each subset V_i consists of several vertices in G. Each vertex in G gives rise

to a complete bipartite subgraph in G^B. Let us consider the set V_i^B which consists of all vertices of all complete bipartite subgraphs corresponding to the vertices in V_i.

It is clear that if $|V_i| \leq W$, then $|V_i^B| \leq 2\times(J+1)\times W = W^B$. It is also clear that the internal edges of the complete bipartite subgraphs do not contribute to the cardinality of the set Q^B, and if any edge e belongs to the subgraph $G(V_i)$ induced by the set of vertices V_i, then the corresponding edge e^B in the graph G^B will belong to the subgraph $G^B(V_i^B)$ induced by the set of vertices V_i^B. This means that $|Q^B| \leq |Q|$ and consequently $|Q^B| \leq J^B$.

Suppose that we have partition $V_1^B, V_2^B, \ldots, V_M^B$ of the set V^B that satisfies the conditions of the bipartite partitioning problem. First, let us note that, because $|Q^B| \leq J^B = J$, the vertices in the graph $G^B(V^B,E^B)$ that belong to the same complete bipartite graph corresponding to a single vertex in $G(V,E)$ must belong to the same set V_i^B. If this is not the case, then according to Lemma 2 the number of edges that connect these vertices and have their endpoints in two different sets of the partition will not be less than $J+1$. This means that $|Q^B|$ will be greater than $J^B = J$, which contradicts our assumption that $|Q^B| \leq J$.

Thus, we have proven that all vertices that belong to the same complete bipartite subgraph, representing some vertex in the graph $G(V,E)$, will belong to the same set V_i^B. This means that each set V_i^B will consist of vertices of several complete bipartite subgraphs, each representing some vertex in $G(V,E)$; or we can say that each set V_i^B is associated with some set V_i of vertices of the graph $G(V,E)$. All such sets $V_1, V_2, \ldots, V_M$ are pairwise disjoint, and their union is equal to the set V. Therefore, the sets $V_1, V_2, \ldots, V_M$ represent a partition of the graph $G(V,E)$.

Now it is evident that if

$$|V_i^B| \leq W^B = 2\times(J+1)\times W \quad \text{for} \quad 1 \leq i \leq M$$

then

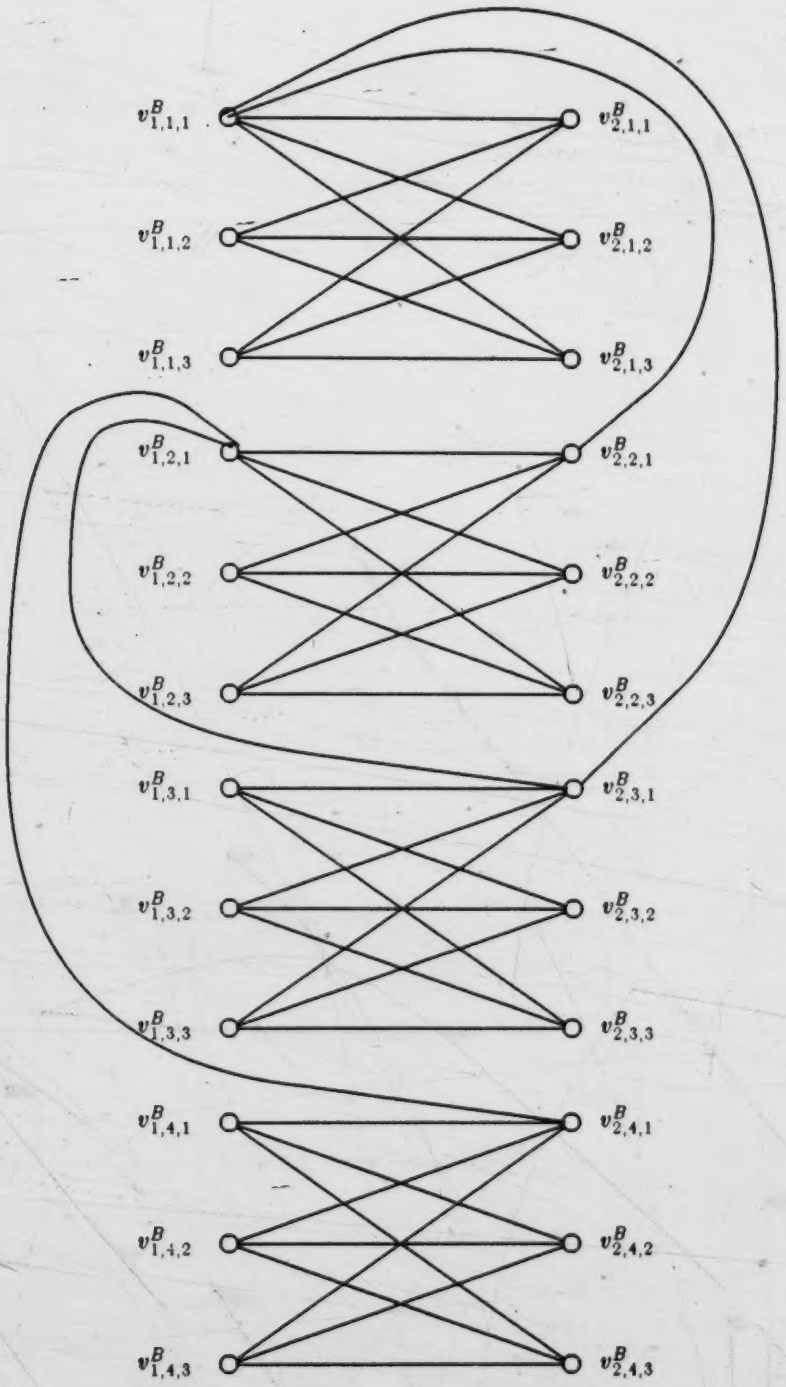

Figure 3.3 Instance of the bipartite partitioning problem

$$|V_i| \leq W \quad \text{for} \quad 1 \leq i \leq M.$$

Also it easy to see that if any edge e^B belongs to the subgraph $G(V_i^B)$ induced by the set of vertices V_i^B, then either this is the internal edge of one of the complete bipartite subgraphs S_i and thus has no corresponding edge in the graph $G(V,E)$, or the corresponding edge e in the graph $G(V,E)$ will belong to the subgraph $G(V_i)$ induced by the set of vertices V_i. This means that $|Q| \leq |Q^B|$ and consequently

$$|Q| \leq J.$$

This concludes the proof of Theorem 4.

3.2. Absolute and ϵ-Approximations to Bipartite Partitioning

In this section we shall present two theorems which are similar to the theorems proven for the general partitioning problem. These theorems state that the problems of finding an absolute approximation and an ϵ-approximation to the bipartite partitioning problem are NP-hard.

Theorem 5. An absolute approximation algorithm to the following problem can exist only if P = NP.

Instance: Bipartite graph $G^B(V^B,E^B)$, positive integer W^B.

Problem: Find a partition of V^B into disjoint sets $V_1^B, V_2^B, \ldots, V_M^B$ such that $|V_i^B| \leq W^B$ for $1 \leq i \leq M$, and such that the cardinality of the set $Q^B \subseteq E^B$ of edges cut by the partition is minimal.

Theorem 6. An ϵ-approximation algorithm with $0 < \epsilon < 1$ to the following problem can exist only if P = NP.

Instance: Bipartite graph $G^B(V^B,E^B)$, weights $w(v) > 0$ for each $v \in V^B$, positive integers W^B and M.

Problem: Find a partition of V^B into no more than M disjoint sets $V_1^B, V_2^B, \ldots, V_M^B$ such that $\sum_{v \in V_i^B} w(v) \leq W^B$ for $1 \leq i \leq M$, and such that the cardinality of the set

$Q^B \subseteq E^B$ of edges cut by the partition is minimal.

Theorems 5 and 6 can be proven in the same way as the corresponding Theorems 2 and 3 for the general partitioning problem. The minor change in the proof of Theorem 5 is that, instead of expanding vertices into complete subgraphs and using Lemma 1, the vertices should be expanded into complete bipartite subgraphs and Lemma 2 should be employed.

Chapter 4

The Tree Partitioning Problem

This part of the thesis is devoted to the design and the analysis of an algorithm for partitioning an acyclic data model into a set of acyclic subject databases.

We shall derive a polynomial time algorithm for partitioning acyclic data models. The algorithm will find an optimal partition of a data model without any restriction on the number of subject databases. We shall also briefly discuss the modifications that can be made to this algorithm in order to find an optimal partition of a data model when there is a bound on the number of subject databases.

4.1. Basic Definitions and Statement of the Problem

In order to make a formal analysis of a partitioning algorithm for acyclic structures, we shall introduce some basic definitions concerning trees.

Definition: A *tree* is a finite set of one or more nodes such that there is a specially designated node called the *root* and the remaining nodes are partitioned into $n \geq 0$ disjoint sets $T_1, T_2, \ldots, T_n$, where each of these sets is a tree. $T_1, T_2, \ldots, T_n$ are called the *subtrees* of the root.

The number of subtrees of a node is called its *degree*. Nodes that have degree zero are called *leaf* or *terminal* nodes. The other nodes are referred to as *nonterminals*. The roots of the subtrees of a node are called the *children* of the node, while the node is called the *parent* of its children. Children of the same parent are called *siblings*. The *degree* of a tree is the maximum degree of the nodes in the tree. The *ancestors* of a node are all the nodes along the path from the root to that node.

The *level* of a node is defined by initially letting the root be at level one. If a node is at level p, then its children are at level $p+1$.

The *height* of the tree is defined to be the maximum level of any node in the tree.

In this chapter we shall consider the partitioning problem where the data model and the transactions have tree-like structures. Namely, we shall assume that all entities are nodes of some data model tree and all transactions can be represented as connected subgraphs of this tree. Subject database are also assumed to be connected subgraphs of the data model tree. Getting a little bit ahead of our discussion, we can say that this assumption implies that our data model can be represented by an *Entity-Relationship Diagram* without *cycles*. A detailed discussion of data modeling using the Entity-Relationship Diagram is presented in the next chapter.

Here, in the same way as for the general partitioning problem, we shall associate nonnegative integer weights with all entities and nonnegative integer costs with all transactions. The optimal partition of the data model tree is the partition having the minimal cost.

In this chapter, we shall introduce a new concept, the *value* of a partition. The value of a partition is the sum of the costs of all transactions that are not cut by the partition. It is easy to see that the partition value plus the partition cost is equal to the sum of the costs of all transactions. It will be more convenient, in this chapter, to use the concept of the partition value instead of the partition cost. Using this new concept we can define an optimal partition as a partition that maximizes the partition value.

Many NP-hard problems on graphs can be solved in polynomial time when graphs are restricted to trees. The partitioning problem happens to be no exception.

The problem of the partitioning trees was first considered by J. A. Lukes[16], who found a polynomial time algorithm for the partitioning of trees without a restriction on the number of clusters. This algorithm returns the optimal partition which minimizes the total cost of all edges cut by the partition.

We shall consider a much more general problem, by assigning costs not only to

edges of a tree, but also to connected subgraphs of the tree. Also, we shall consider the tree partitioning problem with a bound on the number of clusters.

4.2. Tree Partitioning With an Unlimited Number of Clusters

In this section, we shall consider the problem of partitioning a data model tree into several connected subgraphs. First, however, we shall introduce several new definitions. In the definitions that follow, we shall use the notation $T(V,E)$ to refer to the tree T with the set V of nodes and the set E of edges.

Definition: A collection of the trees $T_1(V_1,E_1)$, $T_2(V_2,E_2)$,..., $T_M(V_M,E_M)$ is called a partition of the tree $T(V,E)$ if

$$V_i \cap V_j = \emptyset \quad \text{for} \quad i \neq j, \ 1 \leq i,j \leq M$$

and

$$\bigcup_{1 \leq i \leq M} V_i = V.$$

Very often we shall refer to the trees $T_i(V_i,E_i)$ as *clusters* of the partition.

Definition: A cluster containing the root of the tree $T(V,E)$ is called the *root cluster* of the partition.

Each node v of the tree $T(V,E)$ will be assigned a nonnegative integer weight $w(v)$. We assume that each transaction can be represented by a connected subgraph s of the data model tree. Each transaction subgraph s will be assigned a nonnegative integer cost $l(s)$. A subgraph is said to be cut by the partition if it has nodes in two or more clusters of the partition. The value of the partition is the total sum of the costs of all transactions that are not cut by the partition.

Formally, in terms of trees, the problem of partitioning of an acyclic data model can be formulated as follows:

Instance: Tree $T(V,E)$, where V is the set of vertices and E is the set of edges; set S of connected subgraphs of the tree T; nonnegative integer weights $w(v)$ for each $v \in V$,

and nonnegative integer costs $l(s)$ for each $s \in S$; positive integer W.

Optimization: Find a partition $T_1(V_1,E_1)$, $T_2(V_2,E_2)$,..., $T_M(V_M,E_M)$ of the tree $T(V,E)$ which maximizes the partition value and such that $\sum_{v \in V_i} w(v) \leq W$ for all $1 \leq i \leq M$.

We shall design the tree partitioning algorithm using the dynamic programming technique. In order to formulate the principle of optimality for our problem, we shall enumerate the nodes of the data model tree and introduce some additional concepts and definitions.

For a given tree T with the enumerated nodes, we shall define the connected subgraph $R(v,n)$ which is also a tree, where v is a node of the tree T and n is nonnegative integer. The tree $R(v,n)$ has its root at the node v, and it contains n subtrees whose roots are the first n children of the node v. If $n=0$, then tree $R(v,0)$ contains only one node v. The notation $R(v)$ will stand for the subtree with the root at the node v, so if the node v has only n children, then $R(v,n)=R(v)$.

If we look now at the tree shown in Figure 4.1, then $R(6,0)$ contains only one node (6), $R(5,2)$ contains nodes (5,2,3), and $R(6,1)$ contains nodes (6,1).

Now we are ready to describe informally the steps of the recursive partitioning algorithm using high level routines PARTITION and COMBINE.

Procedure: PARTITION tree T

Step 1. If the tree T is empty, then return.

Step 2. PARTITION all subtrees of the tree T.

Step 3. PARTITION the root of the tree T.

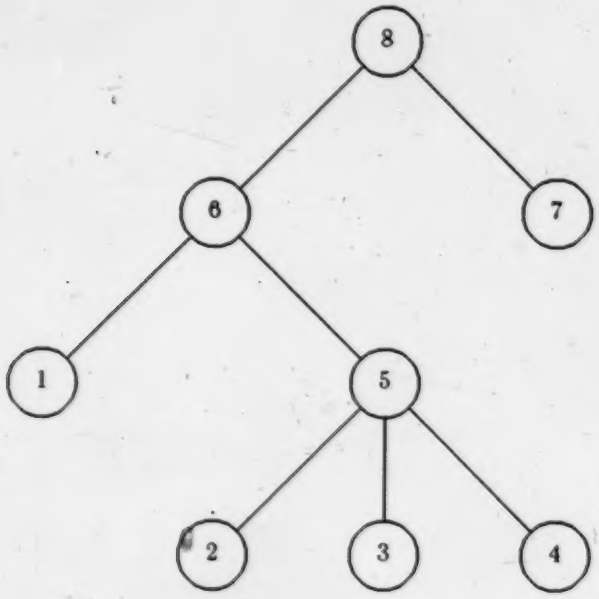

Figure 4.1 Tree with enumerated nodes

// This is a trivial partition of a tree consisting only of one node. This partition will consist of a single cluster containing a single node. //

Step 4. COMBINE partitions of the first subtree and partition of the root.

// This step will return partitions of the tree consisting of the root node of the tree T and its first subtree. //

Step 5. COMBINE partitions of the second subtree with partitions obtained in Step 4.

// This step will return partitions of the tree consisting of the root node of the tree T and its first two subtrees. //

Step 6. COMBINE partitions of the next subtree i with the partitions of the tree consisting of the root node of the tree T and its first $i-1$ subtrees.

// This step will return partitions of the tree consisting of the root node of the tree T and its first i subtrees. //

Step 7. If there are no more subtrees, then return, otherwise go to Step 6.

The routine PARTITION is a recursive procedure which returns partitions of a tree. These partitions will be used to find partitions of the larger trees. For the tree shown in Figure 4.1, we shall find partitions of the trees $R(v,n)$ in the following order:

$R(1,0)$
$R(2,0)$ $R(3,0)$ $R(4,0)$
$R(5,0)$ $R(5,1)$ $R(5,2)$ $R(5,3)$
$R(6,0)$ $R(6,1)$ $R(6,2)$
$R(7,0)$
$R(8,0)$ $R(8,1)$ $R(8,2)$.

The routine COMBINE is used to combine the partitions of a tree consisting of the root node and its first $i-1$ subtrees with the partitions of the $i-th$ subtree to form the partitions of the root node of the tree and its first i subtrees. For the tree shown in Figure 4.2, the routine COMBINE will combine partitions of the tree $R(v,i-1)$ with partitions of the tree $R(v_i)$ to form partitions of the tree $R(v,i)$.

If we have a tree with N nodes, then such a tree will have 2^{N-1} different partitions. This means that if we consider all possible partitions of the tree, then our algorithm will have an exponential time complexity. Therefore, in order to have a polynomial time partitioning algorithm, we should develop some optimality principle which will allow us to discard partitions of the trees $R(v,n)$ which can not be part of the optimal partition of the tree T. The routine COMBINE will apply this principle when forming partitions of the tree $R(v,i)$ from partitions of the trees $R(v,i-1)$ and $R(v_i)$.

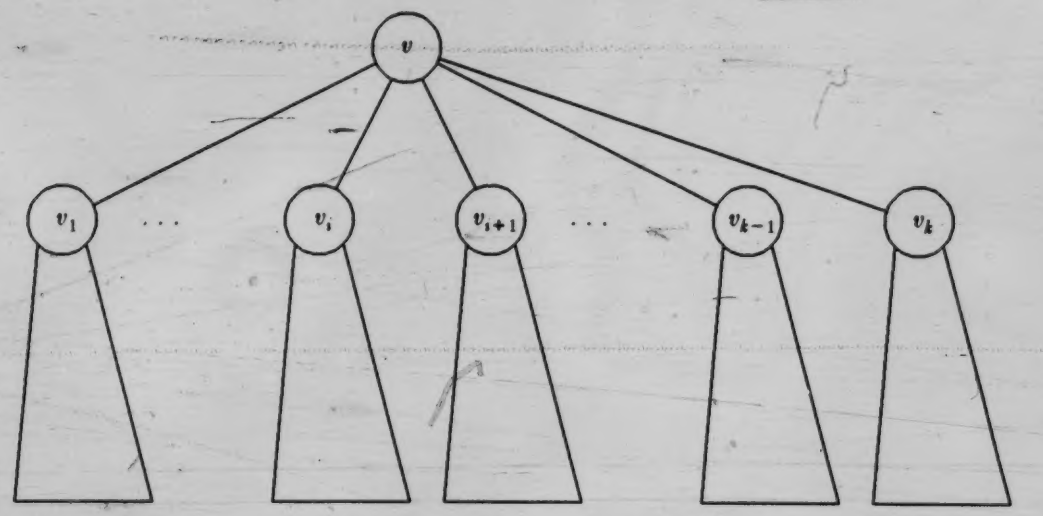

Figure 4.2 Example of the tree $R(v)$

To give the formal description of the partitioning algorithm, we shall introduce several additional definitions. We shall define $S(v,n)$ to be the set of all transaction trees such that $s \in S(v,n)$ iff

$$s \cap R(v,n) \neq \emptyset$$

and

$$s \cap (T - R(v,n)) \neq \emptyset,$$

where T is a data model tree. Here and later, all set-theoretical operations on trees or subgraphs are applied only to the their sets of nodes. This will reduce the number of different variables that have to be introduced.

The definition of $S(v,n)$ implies that this set consists of all transaction trees that have both nodes that belong to the tree $R(v,n)$ and nodes that do not belong to this tree. For the data model tree shown in Figure 4.1 and transaction trees shown in Figure 4.3, the set $S(6,2)$, for example, consists of transactions 1, 2 and 3.

We shall also define the *influence set* of nodes $I(v,n)$ such that

$$I(v,n) = \bigcup_{s \in S(v,n)} s \bigcap R(v,n).$$

According to this definition, the set of nodes $I(v,n)$ consists of all nodes in $R(v,n)$ that are accessed by the transactions from $S(v,n)$. It is clear that $I(v,n)$ is empty or it is a connected tree with the root v, because all transactions are connected subgraphs of the tree T. Set $I(v,n)$ is empty if $R(v,n)$ represents the whole tree T. If set $I(v,n)$ is empty and $R(v,n)$ is a proper subset of the tree T then the tree $R(v,n)$ can be partitioned separately from the rest of the tree T.

We shall define set $Z(q,v,n)$ as the set of all connected subgraphs of the tree $R(v,n)$ such that $z \in Z(q,v,n)$ iff

$$v \in z \quad \text{and} \quad \sum_{u \in z} w(u) = q.$$

That is, $Z(q,v,n)$ is the set of all connected subgraphs of the tree $R(v,n)$ that contain the node v and have weight equal to q.

We shall also define set $A(q,v,n)$ of connected subgraphs such that $a \in A(q,v,n)$ iff there exists $z \in Z(q,v,n)$ such that

$$a = z \bigcap I(v,n).$$

The $i-th$ member of this set will be referred to as $a_i(q,v,n)$. It is clear that $a_i(q,v,n)$ is a tree with a root v.

Now we can define set $B(a_i(q,v,n))$ of all partitions of the tree $R(v,n)$ such that if C is a root cluster of any partition from $B(a_i(q,v,n))$, then

$$\sum_{u \in C} w(u) = q$$

and

$$C \bigcap I(v,n) = a_i(q,v,n).$$

For each set $B(a_i(q,v,n))$ we shall define $P(a_i(q,v,n))$ as a partition from the set

Figure 4.3 Transaction trees

$B(a_i(q,v,n))$ with the maximal value. If there are two or more partitions with the maximal value, then we shall take any one of them. The notation $P(v,n)$ will stand for the optimal partition of the tree $R(v,n)$, and the notation $P(v)$ will stand for the optimal partition of the subtree with the root at the node v.

Finally, we shall define set $Q(v,n)$ which consists of all partitions $P(a_i(q,v,n))$ for given values v and n and all possible values q and i.

If a node v has n children and we want to refer to a tree consisting of the root v and all its n subtrees, then, usually, we drop parameter n. That is,

$$
\begin{aligned}
Z(q,v,n) &= Z(q,v) \\
A(q,v,n) &= A(q,v) \\
a(q,v,n) &= a(q,v) \\
I(v,n) &= I(v) \\
R(v,n) &= R(v) \\
B(a(q,v,n)) &= B(a(q,v)) \\
P(a(q,v,n)) &= P(a(q,v)) \\
P(v,n) &= P(v) \\
Q(v,n) &= Q(v)
\end{aligned}
$$

and so on.

We shall prove that, in order to find an optimal partition of the tree T, we do not have to consider all possible partitions of the trees $R(v,n)$ but only the partitions from the sets $Q(v,n)$.

This fact is at the heart of the designed algorithm. Due to this fact, the majority of partitions of the trees $R(v,n)$ will not be considered at all, because they can not be a part of an optimal partition of the tree T. The routine COMBINE will use this fact. For the tree $R(v,n)$, this routine will return only partitions belonging to the set $Q(v,n)$.

Another important aspect of the routine COMBINE is the way it combines a partition of the tree $R(v,i-1)$ with a partition of the tree $R(v_i)$. This can be achieved by

two basic methods preserving connectivity of the resulting clusters:

1) Concatenate the clusters of two partitions.

2) Merge the root cluster of one partition with the root cluster of another partition and concatenate the resulting cluster with the rest of the clusters from two partitions.

Two routines, CONCATENATE and MERGE, are associated with these two methods.

Now we are ready to present a partitioning algorithm in a formal way.

4.2.1. Statement of the Algorithm

In this section we shall give a formal statement of the algorithm for partitioning of a tree with an unlimited number of clusters, and we shall discuss in detail some parts of the algorithm. We shall use the notation defined previously. The main routine of the partitioning algorithm is the routine PARTITION. The input to this routine is a data model tree and transaction trees, and the output of this routine is the set $Q(v,n)$. The nodes of the tree are referred to as shown in Figure 4.2.

Procedure: PARTITION tree ($R(v)$)

Step 1. If the tree $R(v)$ is empty, then return.

Step 2. For i = 1 to the number of children of the node v, PARTITION tree $R(v_i)$.

// Partition all subtrees of the tree $R(v)$. //

Step 3. PARTITION tree $R(v,0)$.

// This is a trivial partition of the tree $R(v,0)$ which consists of a single cluster containing single node v. This step also creates an influence set $I(v,0)$. The set $I(v,0)$ may be empty or it may contain a single node

v. //

Step 4. $i = 0$.

Step 5. $i = i + 1$.

Step 6. If the tree $R(v)$ has less than i children, then return.

Step 7. Create influence set $I(v,i)$.

Step 8. COMBINE ($Q(v,i-1)$, $Q(v_i)$).

Step 12. Go to Step 5.

The procedure COMBINE is used to combine partitions from the sets $Q(v,i-1)$ and $Q(v_i)$ to obtain the set of partitions $Q(v,i)$. This procedure will use routines CONCATENATE, MERGE and OPTIMALITY-CHECK.

Procedure COMBINE ($Q(v,i-1)$, $Q(v_i)$)

Step 1. Initialize set $Q(v,i)$ to an empty set.

Step 2. For $l = 1$ to W DO

// Create partitions with the weight of the root cluster from 1 to W. //

Step 3 Initialize set $A(l,v,i)$ to an empty set.

Step 4. For all members $a_n(l,v,i-1)$ of the set $A(l,v,i-1)$ DO

Step 5. CONCATENATE ($P(a_n(l,v,i-1))$, $P(v_i)$) giving partition P.

// At this step we concatenate all clusters of the partition $P(a_n(l,v,i-1))$ of the tree $R(v,i-1)$ with all clusters of the partition $P(v_i)$, which is an optimal partition of the tree $R(v_i)$. The weight of the

root cluster of the resulting partition is equal to l. //

Step 6. OPTIMALITY-CHECK of the partition (P).

// This step checks whether the partition P obtained on the previous step should be added to the set $Q(v,i)$, that is, whether it can be part of an optimal partition of the original tree. //

Step 7. Close the loop initiated at Step 4.

Step 8. For $j = 1$ to $l-1$ DO

Step 9. For all members $a_n(j,v,i-1)$ of the set $A(j,v,i-1)$ DO

Step 10. For all members $a_m(l-j,v_i)$ of the set $A(l-j,v_i)$ DO

Step 11. MERGE ($P(a_n(j,v,i-1))$, $P(a_m(l-j, v_i))$) giving partition P.

// At this step we merge the root cluster of the partition $P(a_n(j,v,i-1))$ of the tree $R(v,i-1)$ with the root cluster of the partition $P(a_m(l-j,v_i))$ of the tree $R(v_i)$. The resulting cluster is concatenated with the rest of the clusters of two partitions. The weight of the root cluster of the resulting partition is equal to l.//

Step 12. OPTIMALITY-CHECK of the partition (P).

Step 13. Close loops initiated at Steps 10, 9, 8 and 2.

The procedures CONCATENATE and MERGE are trivial routines explained in the comments to the procedure COMBINE. We should comment, however, on the properties of partitions that are produced by CONCATENATE and MERGE routines.

If partition P is obtained by the CONCATENATE routine applied to the partitions $P(a(l,v,i-1))$ and $P(a(j,v_i))$, then the weight of the root cluster of the partition

P is equal to l and the value of the partition P is equal to the sum of the values of the partitions $P(a(l,v,i-1))$ and $P(a(j,v_i))$.

If, on the other hand, partition P is obtained by the MERGE routine applied to the partitions $P(a(l,v,i-1))$ and $P(a(j,v_i))$, then the weight of the root cluster of the partition P is equal to $l+j$, and the value of the partition P is equal to the sum of values of partitions $P(a(l,v,i-1))$ and $P(a(j,v_i))$ plus the total cost of all transactions that access nodes from both the root clusters of partitions $P(a(l,v,i-1))$ and $P(a(j,v_i))$ and that do not access nodes from any other cluster. Such transactions are cut before merging, but they will be connected by the MERGE operation, thus increasing the total value of the resulting partition.

Now we shall present the description of the routine OPTIMALITY-CHECK.

Procedure: OPTIMALITY-CHECK of partition (P)

Step 1. Calculate the value of the partition P.

Step 2. Set a equal to the intersection of the root cluster of the partition P with the set $I(v,i)$.

Step 3. If a is the $i-$th member of the set $A(l,v,i)$ and value of the partition P is greater than the value of the partition $P(a_i(l,v,i))$ from the set $Q(v,i)$, then replace the partition $P(a_i(l,v,i))$ by the partition P.

Step 4. If $a \notin A(l,v,i)$, then

$m = |A(l,v,i)|+1$

Set $a_m(l,v,i) = a$ and add $a_m(l,v,i)$ to the set $A(l,v,i)$

Set $P(a_m(l,v,i)) = P$ and add $P(a_m(l,v,i))$ to the set $Q(v,i)$.

The routine OPTIMALITY-CHECK is ultimately responsible for discarding partitions which can not be part of the optimal partition of the original tree.

4.2.2. Proof of Optimality of the Partition

We shall prove that the algorithm finds an optimal partition of the data model tree. In this section, we shall use previously-introduced notations and refer to the data model tree shown in Figure 4.2.

The proof of optimality will be preceded by the proof of the validity of the following two statements:

Statement 1. The set of partitions $Q(v,i)$ contains an optimal partition of the tree $R(v,i)$ for all possible values of v and i.

Statement 2. The algorithm finds the set of partitions $Q(v,i)$ correctly by combining partitions from the sets $Q(v,i-1)$ and $Q(v_i)$.

Let us prove the first statement. Let P be an optimal partition of the tree $R(v,i)$ with the partition value V, and let C be a root cluster of this partition such that

$$\sum_{u \in C} w(u) = q$$

$$C \bigcap I(v,i) = a(q,v,i),$$

where $w(u)$ is the weight of the node u. It is clear that the partition P belongs to the set $B(a(q,v,i))$. Partition $P(a(q,v,i))$ is a member of the set $B(a(q,v,i))$, and this partition has the largest value in this set. The value of the partition $P(a(q,v,i))$ is not less than the value of the partition P according to the definition of the partition $P(a(q,v,i))$. If the value of the partition $P(a(q,v,i))$ is greater than the value of the partition P, then partition P can not be the optimal partition of the tree $R(v,i)$. This

means that the value of the partition $P(a(q,v,i))$ is equal to the value of the optimal partition of the tree $R(v,i)$. It is possible, but not necessary, that partitions P and $P(a(q,v,i))$ are the same. What we have shown here is that if partition P is an optimal partition of the tree $R(v,i)$, then the partition $P(a(q,v,i))$ is also an optimal partition of the tree $R(v,i)$. However, partition $P(a(q,v,i))$ belongs to the set $Q(v,i)$, which means that the set $Q(v,i)$ contains the optimal partition of the tree $R(v,i)$.

Now we shall prove the second statement. Let us take an arbitrary tree $R(v,i)$ and the set $Q(v,i)$ associated with this tree. Let us consider any partition $P(a(q,v,i))$ from the set $Q(v,i)$. We shall prove that our algorithm will construct partition P with root cluster C such that

$$\sum_{u \in C} w(u) = q$$

$$C \bigcap I(v,i) = a(q,v,i)$$

and such that the value of the partition P will be equal to the value of the partition $P(a(q,v,i))$. In other words, this means that our algorithm will find either the partition $P(a(q,v,n))$ or some other partition P from the set $B(a(q,v,i))$ having the same partition value. Again, we should note that it is not necessary for the partitions P and $P(a(q,v,i))$ to be equal. We demand only equality of their partition values.

To proceed with the proof, we shall introduce some new notations:

$$S_1 = C \bigcap R(v,i-1)$$

$$S_2 = C \bigcap R(v_i)$$

$$q_1 = \sum_{u \in S_1} w(u)$$

$$q_2 = \sum_{u \in S_2} w(u)$$

$$d_1(q_1,v,i-1) = S_1 \bigcap I(v,i-1)$$

$$d_2(q_2,v_i) = S_2 \bigcap I(v_i).$$

Now we shall make several observations. First, it is easy to see that

$$q_1 + q_2 = q.$$

It follows from the fact that

$$S_1 \cap S_2 = \varnothing \quad \text{and} \quad S_1 \cup S_2 = C.$$

Also, we shall note that

$$I(v,i) \subseteq I(v,i-1) \cup I(v_i).$$

To prove the last statement, let us consider the arbitrary node $u_1 \in I(v,i)$. The fact that u_1 belongs to the set $I(v,i)$ means that there exists node u_2 and transaction t such that

$$u_1 \in R(v,i), \quad u_2 \notin R(v,i), \quad u_1 \in t, \quad u_2 \in t.$$

Taking into account that

$$R(v,i) = R(v,i-1) \cup R(v_i),$$

we can easily obtain that

$$u_2 \notin R(v,i-1)$$
$$u_2 \notin R(v_i)$$
$$u_1 \in R(v,i-1) \cup R(v_i).$$

The last statement means that either $u_1 \in R(v,i-1)$ or $u_1 \in R(v_i)$. If u_1 belongs to $R(v,i-1)$, then according to the definition of the set $I(v,i-1)$, we have that

$$u_1 \in I(v,i-1).$$

If u_1 belongs to $R(v_i)$, then according to the definition of the set $I(v_i)$, we have that

$$u_1 \in I(v_i).$$

It follows from the last two statements that

$$u_1 \in I(v,i-1) \cup I(v_i),$$

which leads to the conclusion that

$$I(v,i) \subseteq I(v,i-1) \cup I(v_i).$$

Also it is clear that $d_1 \in A(q_1,v,i-1)$ and $d_2 \in A(q_2,v_i)$.

Now let us consider two cases:

Case 1. $S_2 = \varnothing$.

Case 2. $S_2 \neq \varnothing$.

The first case means that

$$q_1 = q \text{ and } q_2 = 0$$

and that the partition $P(a(q,v,i))$ is the result of the concatenation of clusters of some partition P_1 of the tree $R(v,i-1)$ and of clusters of some partition P_2 of the tree $R(v_i)$. Let C_1 be the root cluster of the partition P_1. Then we can easily see that $C_1 = C = S_1$, and as a result we have

$$\sum_{u \in C_1} w(u) = \sum_{u \in C} w(u) = q$$

$$C_1 \bigcap I(v,i-1) = S_1 \bigcap I(v,i-1) = d_1(q,v,i-1).$$

We insist now that the partition P_1 have the same value as a partition $P(d_1(q,v,i-1))$ from the the set $Q(v,i-1)$ and that the partition P_2 be the optimal partition of the tree $R(v_i)$.

Really, both partitions P_1 and $P(d_1(q,v,i-1))$ belong to the set $B(d_1(q,v,i-1))$. By definition, the value of the partition $P(d_1(q,v,i-1))$ can not be less than the value of the partition P_1. On the other hand, if the value of the partition $P(d_1(q,v,i-1))$ is greater than the value of the partition P_1, then by concatenating clusters of the partitions $P(d_1(q,v,i-1))$ and P_2, we shall obtain partition $P^* \in B(a(q,v,i))$ whose value will be greater than the value of the partition $P(a(q,v,i))$. This, however, contradicts the definition of the partition $P(a(q,v,i))$. Thus, we have proven that the partitions P_1 and $P(d_1(q,v,i-1))$ have the same value.

Now we shall prove that the partition P_2 is the optimal partition of the tree $R(v_i)$. If partition P_2 is not optimal, then we can replace it by the optimal partition of the tree $R(v_i)$, and concatenate clusters of this optimal partition with the clusters of the partition P_1. The resulting partition of the tree will belong to the set $B(a(q,v,i))$,

and the value of this partition will be greater than the value of the partition $P(a(q,v,i))$. This, however, contradicts the definition of the partition $P(a(q,v,i))$.

Thus, we have proven that in the case of $S_2 = \varnothing$, the partition $P \in B(a(q,v,i))$ having the same value as the partition $P(a(q,v,i)) \in Q(v,i)$ can be constructed by concatenating clusters of the optimal partition of the tree $R(v_i)$ with the clusters of the partition $P(d_1(q,v,i-1))$. Set $Q(v_i)$ contains the optimal partition of the tree $R(v_i)$, and partition $P(d_1(q,v,i-1))$ belongs to the set $Q(v,i-1)$.

Now we shall consider the second case, when $S_2 \neq \varnothing$. In this case, the partition $P(a(q,v,i))$ consists of three types of clusters:

Type 1. Clusters that are contained entirely in the tree $R(v,i-1)$.

Type 2. Clusters that are contained entirely in the tree $R(v_i)$.

Type 3. Root cluster as the only cluster having nodes from both trees $R(v,i-1)$ and $R(v_i)$.

Let us consider partition P_1 of the tree $R(v,i-1)$, which consists of all first type clusters of the partition $P(a(q,v,i))$ concatenated with the cluster S_1, which will be the root cluster of this partition. We shall also consider partition P_2 of the tree $R(v_i)$, which consists of all second type clusters of the partition $P(a(q,v,i))$ concatenated with the cluster S_2, which will be the root cluster of this partition. It is obvious that $P_1 \in B(d_1(q_1,v,i-1))$ and $P_2 \in B(d_2(q_2,v_i))$.

The partition $P(a(q,v,i))$ can be constructed from the partitions P_1 and P_2 with the help of the MERGE routine by merging root clusters S_1 and S_2 of partitions P_1 and P_2 and concatenating the resulting cluster with the rest of the clusters from the two partitions.

Let P^* be the partition which is constructed by applying the MERGE operation to the partitions $P(d_1(q_1,v,i-1))$ and $P(d_2(q_2,v_i))$. We shall prove that the partition

P^* belongs to the set $B(a(q,v,i))$ and that the value of the partition P^* is equal to the value of the partition $P(a(q,v,i))$. To proceed with the proof, we shall introduce the following notations:

C^*_1 is root cluster of the partition $P(d_1(q_1,v,i-1))$,

C^*_2 is root cluster of the partition $P(d_2(q_2,v_i))$,

C^* is root cluster of the partition P^*.

Using these new notations, we have

$$\sum_{u \in C^*_1} w(u) = q_1$$

$$\sum_{u \in C^*_2} w(u) = q_2$$

$$C^* = C^*_1 \bigcup C^*_2$$

$$C^*_1 \bigcap C^*_2 = \varnothing$$

$$d_1(q_1,v,i-1) = C^*_1 \bigcap I(v,i-1)$$

$$d_2(q_2,v_i) = C^*_2 \bigcap I(v_i).$$

In order to prove that $P^* \in B(a(q,v,i))$, we must prove that

$$\sum_{u \in C^*} w(u) = q$$

and

$$C^* \bigcap I(v,i) = a(q,v,i).$$

The proof of the first equality follows from the fact that C^* is a union of two disjoint sets C^*_1 and C^*_2 so that we have

$$\sum_{u \in C^*} w(u) = \sum_{u \in C^*_1} w(u) + \sum_{u \in C^*_2} w(u) = q_1 + q_2 = q.$$

Now we shall prove the second equality. From the fact that

$$I(v,i) \subseteq I(v,i-1) \bigcup I(v_i),$$

which was proven earlier, it follows that

$$I(v,i) \bigcap (I(v,i-1) \bigcup I(v_i)) = I(v,i).$$

We also know that

$$C^*_1 \bigcap I(v_i) = \emptyset \quad \text{and} \quad C^*_2 \bigcap I(v,i-1) = \emptyset.$$

Now we can easily derive that

$$\begin{aligned} C^* \bigcap I(v,i) &= (C^*_1 \bigcup C^*_2) \bigcap I(v,i) \\ &= (C^*_1 \bigcup C^*_2) \bigcap (I(v,i) \bigcap (I(v,i-1) \bigcup I(v_i))) \\ &= (C^*_1 \bigcup C^*_2) \bigcap (I(v,i-1) \bigcup I(v_i)) \bigcap I(v,i) \\ &= (C^*_1 \bigcap I(v,i-1) \bigcup C^*_2 \bigcap I(v_i)) \bigcap I(v,i) \\ &= (d_1(q_1,v,i-1) \bigcup d_2(q_2,v_i)) \bigcap I(v,i). \end{aligned}$$

According to the definition of the sets $d_1(q_1,v,i-1)$ and $d_2(q_2,v_i)$, we have

$$\begin{aligned} d_1(q_1,v,i-1) \bigcup d_2(q_2,v_i) &= S_1 \bigcap I(v,i-1) \bigcup S_2 \bigcap I(v_i) \\ &= C \bigcap R(v,i-1) \bigcap I(v,i-1) \bigcup C \bigcap R(v_i) \bigcap I(v_i) \\ &= C \bigcap (I(v,i-1) \bigcup I(v_i)). \end{aligned}$$

In deriving the last equality, we have used the fact that

$$I(v,i-1) \bigcap R(v,i-1) = I(v,i-1) \quad \text{and} \quad I(v_i) \bigcap R(v_i) = I(v_i).$$

Finally, we have

$$\begin{aligned} C^* \bigcap I(v,i) &= (d_1(q_1,v,i-1) \bigcup d_2(q_2,v_i)) \bigcap I(v,i) \\ &= C \bigcap (I(v,i-1) \bigcup I(v_i)) \bigcap I(v,i) \\ &= C \bigcap I(v,i) \\ &= a(q,v,i). \end{aligned}$$

This concludes the proof that $P^* \in B(a(q,v,i))$. The fact that $P^* \in B(a(q,v,i))$ means that the value of the partition P^* cannot be greater than the value of the partition $P(a(q,v,i))$, according to the definition of the partition $P(a(q,v,i))$.

Now we shall prove that the value of the partition P^* is not less than the value of the partition $P(a(q,v,i))$. First, let us consider the set of transactions F such that $t \in F$ iff

$$t \bigcap S_1 \neq \emptyset, \quad t \bigcap S_2 \neq \emptyset, \quad \text{and} \quad t \subseteq S_1 \bigcup S_2 = C$$

and the set of transaction F^* such that $t \in F^*$ iff

$$t \cap C^*_1 \neq \emptyset, \quad t \cap C^*_2 \neq \emptyset, \quad \text{and} \quad t \subseteq C^*_1 \cup C^*_2 = C^*.$$

Let us prove that $F \subseteq F^*$. Let us take arbitrary transaction t such that $t \in F$. According to the definition of the sets $I(v,i-1)$ and $I(v_i)$ and due to the fact that $t \in F$, we have

$$t \cap I(v,i-1) \neq \emptyset,$$
$$t \cap I(v_i) \neq \emptyset, \quad \text{and}$$
$$t \subseteq I(v,i-1) \cup I(v_i).$$

We also know that

$$t \subseteq S_1 \cup S_2 = C.$$

From the last two statements, we can easily derive that

$$t \subseteq (I(v,i-1) \cup I(v_i)) \cap (S_1 \cup S_2) = I(v,i-1) \cap S_1 \cup I(v_i) \cap S_2$$
$$= d_1(q_1,v,i-1) \cup d_2(q_2,v_i).$$

We can also see that

$$t \cap I(v,i-1) = t \cap (S_1 \cup S_2) \cap I(v,i-1)$$
$$= t \cap S_1 \cap I(v,i-1)$$
$$= t \cap d_1(q_1,v,i-1)$$

which means that

$$t \cap d_1(q_1,v,i-1) \neq \emptyset.$$

In the same way we can show that

$$t \cap d_2(q_2,v_i) \neq \emptyset.$$

However, it is obvious that

$$d_1(q_1,v,i-1) \subseteq C^*_1, \quad \text{and} \quad d_2(q_2,v_i) \subseteq C^*_2,$$

which means that

$$t \cap C^*_1 \neq \emptyset, \quad t \cap C^*_2 \neq \emptyset, \quad \text{and} \quad t \subseteq C^*_1 \cup C^*_2 = C^*.$$

According to the definition of the set F^*, this means that $t \in F^*$. Thus, we have proven that if $t \in F$, then $t \in F^*$; in other words, that $F \subseteq F^*$.

The value V of the partition $P(a(q,v,i))$ is equal to

$$V = V_1 + V_2 + V_3$$

where

V_1 is the value of the partition P_1,

V_2 is the value of the partition P_2, and

V_3 is the total cost of all transactions from the set F.

On the other hand, the value V^* of the partition P^* is equal to

$$V^* = V^*_1 + V^*_2 + V^*_3$$

where

V^*_1 is the value of the partition $P(d_1(q_1,v,i-1))$,

V^*_2 is the value of the partition $P(d_2(q_2,v_i))$, and

V^*_3 is the total cost of all transactions from the set F^*.

Both partitions P_1 and $P(d_1(q_1,v,i-1))$ belong to the same set $B(d_1(q_1,v,i-1))$. This means that the value of the partition P_1 cannot be greater than the value of the partition $P(d_1(q_1,v,i-1))$; in other words

$$V^*_1 \geq V_1.$$

In the same way we have

$$V^*_2 \geq V_2.$$

Due to the fact that $F \subseteq F^*$, we have that

$$V^*_3 \geq V_3.$$

All of this finally means that the value of the partition P^* is not less then the value of the partition $P(a(q,v,i))$. We have proven earlier that the value of the partition P^* is not greater that the value of the partition $P(a(q,v,i))$. This means that the value of the partition P^* is equal to the value of the partition $P(a(q,v,i))$.

Summarizing what we have proven so far, we can state the following. Let partition $P(a(q,v,i))$ belong to the set $Q(v,i)$ and have value V. Then we can construct partition $P^* \in B(a(q,v,i))$, having the same value as the partition $P(a(q,v,i))$. The partition P^* can be constructed by applying the operation CONCATENATE to some parti-

tion $P(d_1(q_1,v,i-1)) \in Q(v,i-1)$ and an optimal partition $P(v_1) \in Q(v_1)$ of the tree $R(v_1)$, or by applying the operation MERGE to some partitions $P(d_1(q_1,v,i-1)) \in Q(v,i-1))$ and $P(d_2(q_2,v_1)) \in Q(v_1)$. It is clear that the partition P^* can be a member of the set $Q(v,i)$ instead of the partition $P(a(q,v,i))$.

Now let us look at the procedure COMBINE. Step 5, which is inside the loops initiated in Steps 2 and 4, concatenates all partitions from the set $Q(v,i-1)$ with the optimal partition of the tree $R(v_1)$. Step 11, which is inside the loops initiated in Steps 2, 8, 9, and 10, merges each partition from the set $Q(v,i-1)$ with such partitions from the set $Q(v_1)$ that the weight of the root cluster of the resulting partition is not greater than W. It is clear now that the procedure COMBINE will pick up partitions P_1 and P_2, and will either concatenate or merge them to produce the corresponding partition $P \in B(a(q,v,i))$ with the partition value equal to V. This will insure that the set $Q(v,i)$ will contain partition $P \in B(a(q,v,i))$, such that the partition P has the largest partition value in the set $B(a(q,v,i))$. This concludes the proof of the fact that the set of the partitions $Q(v,i)$ will be created correctly from the sets $Q(v,i-1)$ and $Q(v_1)$.

To finish the proof of the optimality, we should observe that our algorithm always starts from the sets Q containing only trivial partitions of single nodes and proceeds until the whole tree is partitioned. This means that we start with the valid sets Q of partitions, which in turn, means that we shall have the valid set Q for each tree R. The set $Q(v)$ for the whole tree $R(v)$ will contain the optimal partition.

4.2.3. Complexity of the Partitioning Algorithm

This section is devoted to the analysis of the complexity of the tree partitioning algorithm. In this section, we shall use the previously defined notations and shall also introduce some new variables. First, let us consider the input to our algorithm:

Tree R with set V of nodes and set E of edges.

Weights $w(v)$ for each node $v \in V$.

Set T of transaction trees.

Cost $l(t)$ for each transaction $t \in T$.

Bound W on the weights of partition clusters.

Some new variables, which will be very important for the analysis of the algorithm time complexity, will be introduced:

N - number of nodes in the tree.

D - degree of the tree.

$h(t)$ - number of edges in the transaction tree $t \in T$.

$H = \max(h(t))$ for $t \in T$ - maximal number of edges in transaction tree.

$B = |T|$ - number of transactions.

The time complexity of our algorithm is a very complex function of the variables N, B, D, H. So, in order to make the analysis feasible, we shall make some assumptions about the values of these variables. We shall assume that the values of these variables satisfy the following inequalities:

$$B > N, \quad N >> W, \quad W >> D, \quad W >> H.$$

We shall also assume that D and H are constant. These assumptions are very reasonable for the data modelling environment.

In carrying out our analysis, we shall assume that the weights of all nodes are equal to one. In this case, we must consider only transactions such that

$$h(t) < W,$$

because if $h(t) \geq W$ then such transaction t can not fit any cluster and will be always cut by any partition.

We shall also assume that all transactions are evenly distributed throughout the data model tree. Under this condition, we can say that each node is accessed by $O(\frac{BH}{N})$ transactions.

Before we proceed with the analysis of specific routines, we should estimate the cardinality of the sets $A(q,v_i)$ and $A(q,v,i-1)$, which are used by the majority of the routines of the algorithm.

According to the definition of the set $A(q,v_i)$, a tree $a \in A(q,v_i)$ iff there exists a tree $T(V,E)$ such that

1. $v_i \in V$,
2. $T(V,E)$ is a connected subgraph of the tree $R(v_i)$,
3. $|V| = q$, and
4. $I(v_i) \bigcap T(V,E) = a$.

From this definition it is clear that the cardinality of the set $A(q,v_i)$ can not be greater than the number of connected subgraphs of the tree $I(v_i)$. For the given values of D and H, the number of different structures of the tree $I(v_i)$ is finite. This means that the maximal number of connected subgraphs that the tree $I(v_i)$ can contain depends only on the values of D and H, which are constant. The same is true about the set $A(q,v,i-1)$. This means that there exists a constant f, such that for all possible values of q, v and i we have

$$|A(q,v_i)| \leq f \quad \text{and} \quad |A(q,v,i-1)| \leq f.$$

Our partitioning algorithm will use the linked list data structure shown in Figure 4.4 in order to store the structure of a single partition. Each box in Figure 4.4

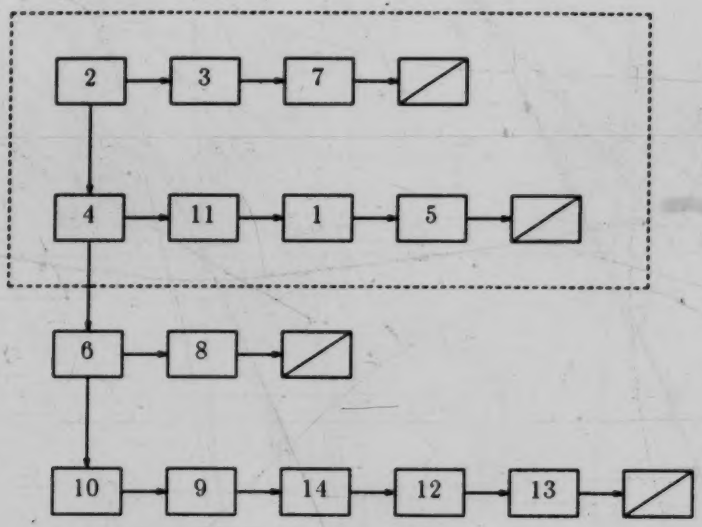

Figure 4.4 Data structure to store partition

corresponds to a single node of a tree. The first two lists of boxes correspond to the root cluster of the partition. The first of these two lists represents the intersection of the root cluster with the corresponding influence set of nodes. So, if we have some partition $P(a(q,v,i))$, then the first list represents the set $a(q,v,i)$. The reason for splitting the root cluster into two parts will be explained below. Each of the rest of the lists corresponds to a single cluster. The figure shows an example of a partition of a tree with 14 nodes. The root cluster of this partition is (2,3,7,4,11,1,5), and the rest of the clusters are (6,8), (10,9,14,12,13).

The two procedures that will create these data structures are CONCATENATE and MERGE. Procedure CONCATENATE ($P(a(l,v,i-1))$, $P(a(j,v_i))$) will merge the first two lists of the data structure representing the partition $P(a(j,v_i))$ into a single list representing the root cluster of this partition, and then it will link the first node of this list to the first node of the last list of the data structure representing the partition

$P(a(l,v,i-1))$.

Thus, the time complexity of the CONCATENATE routine is

$$X_{CONC} = O(1).$$

The procedure MERGE ($P(a(l,v,i-1))$, $P(a(j,v_1))$) will merge the first list of the nodes of the partition $P(a(l,v,i-1))$ with the first list of the nodes of the partition $P(a(j,v_1))$, that is the sets $a(l,v,i-1)$ and $a(j,v_1)$. The intersection of the resulting set with the influence set $I(v,i)$ will be the first list in the data structure representing the resulting partition. The nodes that belong to the set

$$a(l,v,i-1) \bigcup a(j,v_1) - I(v,i)$$

will be merged with the second list of the partition $P(a(l,v,i-1))$ and with the second list of the partition $P(a(j,v_1))$. The resulting set will become the second list in the new partition. The rest of the lists of two partitions will be linked together. Thus, the time complexity of the MERGE routine is

$$X_{MERGE} = O(1).$$

Now let us consider time complexity X_{OPT} of the procedure OPTIMALITY-CHECK.

In Step 1 we calculate the value of a partition P of the tree $R(v,i)$. The partition P is the result of combining the partition $P(a(l,v,i-1))$ of the tree $R(v,i-1)$ with the partition $P(a(l-j,v_1))$ of the tree $R(v_1)$. The value of the partition P is equal to the value of the partition $P(a(l,v,i-1))$ plus the value of the partition $P(a(l-j,v_1))$ and the total cost of all transactions that are not totally contained either in the root cluster of the partition $P(a(l,v,i-1))$, or in the root cluster of the partition $P(a(l-j,v_1))$ but are totally contained in the root cluster of the partition P. If C_1 is the root cluster of the partition $P(a(l,v,i-1))$ and C_2 is the root cluster of the partition $P(a(l-j,v_1))$, then such transactions that additionally contribute to the value of the partition P

satisfy the following conditions:

$$t \cap C_1 \neq \emptyset;\ t \cap C_2 \neq \emptyset;\ t \subseteq C_1 \cup C_2.$$

To check these conditions will require

$$|t|\ln|C_1| + |t|\ln|C_2| + |t|\ln|C_1 \cup C_2| = O(H \ln W) = O(\ln W)$$

units of time. It is easy to see, however, that these conditions are equivalent to the following conditions:

$$t \cap a(l,v,i-1) \neq \emptyset;\ \ t \cap a(l-j,v_i) \neq \emptyset;\ \ t \subseteq a(l,v,i-1) \cup a(l-j,v_i).$$

The time required to check these conditions depends only on the sizes of the sets $a(l,v,i-1)$, $a(l-j,v_i)$ and t. However, if D and H are constant, then the sizes of these sets will be limited by constants. So it is possible to check these conditions in $O(1)$ units of time. This was the reason why we had to split the root cluster of the partitions into two parts. It is clear that all transactions satisfying these conditions should access node v, which is the root of the tree $R(v,i)$. The number of such transactions is $O(\frac{BH}{N})$. This means that the time complexity of Step 1 is $O(B\frac{H}{N})$.

The time complexity of Step 2 is $O(1)$. If partition P with the root cluster C is the result of combining the partitions $P(a(l,v,i-1))$ and $P(a(l-j,v_i))$, then

$$I(v,i) \cap C = I(v,i) \cap (a(l,v,i-1) \cup a(l-j,v_i)).$$

This means that it will take $O(1)$ units of time to find the intersection of the set $I(v,i)$ with the root cluster C.

The time complexity of Step 3 and Step 4 is $O(1)$. Finally, the time complexity X_{OPT} of the routine OPTIMALITY-CHECK is

$$X_{OPT} = O(\frac{B}{N}H) = O(\frac{BH}{N}).$$

Now we shall consider the time complexity X_{COMB} of the procedure COMBINE applied to the sets $Q(v,i-1)$ and $Q(v_i)$. Let f_{CONC} be the number of CONCATENATE operations, and f_{MERGE} be the number of MERGE operations performed by the

procedure COMBINE. The time complexity X_{COMB} can be calculated as follows:

$$X_{COMB} = O(f_{CONC}(X_{CONC} + X_{OPT}) + f_{MERGE}(X_{MERGE} + X_{OPT})).$$

Now, taking into account that $f_{CONC} = O(f_{MERGE})$, we can simplify the expression for X_{COMB} as follows:

$$X_{COMB} = O(f_{MERGE}(X_{MERGE} + X_{OPT})).$$

However,

$$X_{OPT} + X_{MERGE} = O(\frac{B}{N}) + O(1) = O(\frac{B}{N}).$$

And finally we have

$$X_{COMB} = f_{MERGE}\ O(\frac{B}{N}).$$

The MERGE routine is located in the procedure COMBINE inside the loops initiated in Steps 2, 8, 9, and 10. This means that the value f_{MERGE} can be calculated as follows:

$$f_{MERGE} = \sum_{l=1}^{W} \sum_{j=1}^{l-1} |A(j,v,i-1)|\ |A(l-j,v_i)|.$$

Now we can get the following estimate for the number of MERGE operations:

$$f_{MERGE} \le \sum_{l=1}^{W} \sum_{j=1}^{l-1} f^2 = f^2 \sum_{l=1}^{W} \sum_{j=1}^{l-1} 1.$$

The last expression shows that

$$f_{MERGE} = f^2\ O(W^2).$$

Now we can have the following expression for the complexity of the COMBINE procedure:

$$X_{COMB} = O(f^2 \frac{B}{N} W^2).$$

Now let us consider the complexity of the procedure PARTITION. The procedure PARTITION will apply $N-1$ times the procedure COMBINE to the tree being partitioned. This means that the time complexity X_{PART} of the procedure PARTITION is

$$X_{PART} = O(N\, X_{COMB})$$

or, taking into account the expression for X_{COMB}, we can have that

$$X_{PART} = f^2 O(BW^2).$$

The expression for X_{PART} shows that if the degree D of the tree, the maximal number H of nodes accessed by the transaction, and the maximal size W of clusters are limited, then the time complexity of the partitioning algorithm will be linear with respect to the number of transactions. This is the same in data modelling, where the values W, D and H are limited, but the numbers N and B can be relatively large.

We should note however, that the constant f can be very large if the values of the variables D and H are large. Consider, for example, a tree $I(v_i)$ such that each node of this tree has degree D, and each leaf node of this tree has the same depth $H-1$. The number of leaf nodes of this tree is D^{H-1}. This means that the number of different connected subgraphs of this tree is greater than $2^{D^{H-1}}$. So, if $W >> D^{H-1}$, then the cardinality of the sets will grow at least as fast as $2^{D^{H-1}}$. This means that the algorithm will have practical use only for

$$D \leq 3 \quad \text{and} \quad H \leq 4.$$

Appendix 2 contains an example of applying this algorithm to partition the tree.

4.3. Tree Partitioning With a Limited Number of Clusters

In this section, we shall briefly discuss modifications that should be done to the designed partitioning algorithm in order to find the optimal partition of the tree with a limited number of clusters. We shall redefine some of the sets defined in section 4.2.

We shall define set $Z(k,q,v,n)$ as the set of all connected subgraphs of the tree $R(v,n)$ such that $z \in Z(k,q,v,n)$ iff

1. There exists partition P of the tree $R(v,i)$ that contains exactly k clusters, each having a weight less than W.

2. z is the root cluster of the partition P.

3. The weight of the root cluster z is equal to q.

We shall define the set $A(k,q,v,n)$ as the set of all connected subgraphs such that $a \in A(k,q,v,n)$ iff there exists $z \in Z(k,q,v,n)$ such that

$$a = z \bigcap I(v,n),$$

where $I(v,n)$ is the influence set of the tree $R(v,n)$. The $i-th$ member of this set will be referred to as $a_i(k,q,v,n)$.

We shall define the set $B(a_i(k,q,v,n))$ of all partitions of the tree $R(v,n)$ such that the partition P with the root cluster C belongs to the set $B(a_i(k,q,v,n))$ iff

1. Partition P has exactly k clusters, each satisfying the weight constraint.

2. The weight of the root cluster of the partition P is equal to q.

3. $C \bigcap I(v,n) = a_i(k,q,v,n)$.

For each set $B(a_i(k,q,v,n))$, we shall define partition $P(a_i(k,q,v,n))$ as a partition from the set $B(a_i(k,q,v,n))$ having the maximal value.

Finally, we shall define set $Q(v,n)$ which consists of all partitions $P(a_i(k,q,v,n))$ for a given v and n and for all possible combinations of k and q.

The set $Q(v,n)$ plays the same role here as in partitioning a data model tree with an unlimited number of clusters. That is, for each tree $R(v,n)$ we should consider only those partitions that belong to the set $Q(v,n)$.

The only change to the algorithm will be in the routine COMBINE. We shall have to add another loop to this routine. Inside this loop, the algorithm will generate partitions with a different number of clusters. The number of clusters k will be treated in the same way as the weight of the root cluster q.

Chapter 5

Review of Database Design Methods

In the present chapter we shall informally discuss the different stages of the database design process, and the place where our partitioning algorithm fits in.

5.1. Database Design Objectives

One can define many different database design objectives; however, finally, all of them can be reduced to the following: minimize the total cost of *maintenance* and *operations*.

In the modern environment, maintenance cost is primarily determined by the salaries and wages paid to the system analysts, while operations cost is determined by the cost of CPU cycles and by the number of I/O operations. During the last 15-20 years, we have been witnessing the drastic increase in the cost of analysts' time and the drastic decrease in the cost of CPU time and peripherals usage. There is little doubt that this trend will continue.

As we have already discussed in the Introduction, maintenance cost depends on the way we partition the application system into subject databases and subsystems. This means that maintenance cost will be determined mainly by high-level logical design. Operations cost, on the other hand, depends strongly on the physical design of a database.

There is certainly a trade-off between low maintenance cost and low operations cost. Some decisions taken during high-level logical design in an attempt to reduce maintenance cost can lead to an increase in operations cost. This means that, in order to minimize total maintenance and operations cost, the issues of physical database design should be addressed in the early stages of database design. This approach, however, is not feasible from a theoretical point of view, because there will be too many different parameters and constraints to be considered simultaneously. Also, this

approach is absolutely impossible to implement, because during the early stages of design we just do not know all the parameters which are essential to the physical database design. Taking into account that the maintenance-related costs are constantly growing while operations-related costs are constantly decreasing, one can suggest the following practical approach to database design:

1. Establish a high-level logical design which insures low maintenance cost, and
2. within the established logical framework, carry out the physical database design with the aim of minimizing the operational cost.

This means that, at first, we shall partition the data model into subject databases, taking into account only maintenance issues, and then we shall carry out physical database design for each subject database.

Sometimes, however, the physical design issues should be addressed during high-level design. It is possible, for example, that two entities participate only in simple (low complexity cost) transactions, but some of these transactions require very small response time. This means that the records representing these two entities should be placed physically close to each other in the external memory. If, however, these entities are placed in different subject databases during high-level design, then this will be impossible to correct during the physical database design. So, in order to insure that these two entities are placed in the same subject database, we can introduce a dummy high-cost transaction accessing these two entities.

5.2. Inputs to Database Design

The inputs to the database design consist of a set of entities with their associated weights, a set of relationships between the entities, and a set of transactions with their associated costs (or weights). Basically, there are two types of relationships between entities:

1. a *one-to-one* relationship between entities A and B, which means that one instance of entity A is associated with no more than one instance of entity B, and
2. a *one-to-many* relationship between entities A and B, which means that one instance of entity A can be associated with zero, one, or many instances of entity B.

We shall not consider many-to-many relationships, because they can be reduced to two one-to-many relationships. Also, we assume that relationships involving several entities are reduced to several one-to-many relationships, each involving two entities. The entities and their relationships are usually represented by a so-called Entity-Relationship diagram.

5.2.1. The Entity-Relationship Diagram

An Entity-Relationship (ER) diagram is a graph $G(V,E)$ in which each node corresponds to an entity and each edge corresponds to a relationship between connected entities. The graph $G(V,E)$ can have directed and undirected edges. An undirected edge represents a one-to-one relationship between the connected entities. A directed edge (A,B) represents a one-to-many relationship between entities A and B. Due to the fact that there can be several relationships between two entities, multiple edges can connect the corresponding nodes of the graph.

Each edge of the ER diagram, directed or undirected, also represents link or path

that is used by transactions to navigate through the database.

5.2.2. Structure of a Transaction

Each transaction accesses a specific subset of entities by navigating through the links of the Entity-Relationship diagram. This means that each transaction will have associated with it a graph $G_t(V_t,E_t)$ such that

$$V_t \subseteq V \quad \text{and} \quad E_t \subseteq E$$

where $G(V,E)$ is the graph representing the ER diagram. We should note, however, that the transaction graph $G_t(V_t,E_t)$ can differ from the subgraph of the graph $G(V,E)$ induced by the set of nodes V_t.

Each transaction can be associated with two weights: the first weight reflects complexity of the transaction, and the second weight reflects frequency of the transaction. The complexity of the transaction is important during the high-level logical design when the data model is partitioned into subject data bases. The frequences of the transactions are more important during physical database design.

The complexity weight is assigned to a transaction as a whole, and if any transaction edge is cut by the partition, then it is considered to be an additional overhead because this transaction will be spread through different subsystems and will have to be maintained jointly by two or more maintenance teams. However, not all entities accessed by the transaction play the same role during transaction processing. Only a small subset of all entities accessed by the transaction will participate in complex interrelated data manipulations. The rest of the entities will play only an auxiliary role during transaction processing and they will serve mainly for code expansion. If we consider, for example, a one-to-many relationship between an employee and a department, then the employee record will usually contain a department code. Some transactions will access the department record only to get the name of the department with a given code. If one such transaction is cut, such that the department entity is

in one subject data base and the rest of the entities accessed by this transaction are in another subject data base, then it will hardly increase the cost of system maintenance, because as far as this transaction is concerned, the department entity is very loosely related to the rest of entities accessed by this transaction. Certainly, there can be other transactions for which this link is very important, and they will perform a lot of data manipulations involving attributes of both the department entity and the employee entity. If such a transaction is cut through this link, then it will definitely increase maintenance overhead.

Practical observation is such that usually one half of all entities accessed by a transaction actively participate in the interrelated calculations, while the other half is only loosely related to the rest of entities accessed by the transaction. If we look at the transaction graph, then these loosely related entities will have no more than one edge adjacent to it. Another practical observation is that a transaction graph is usually acyclic, that is a tree, and the loosely connected entities are usually leaves of the transaction tree. These facts make it much easier to apply the developed tree partitioning algorithm to real life problems. Each transaction will access at most 5-6 entities, of which 2-3 will be loosely related to the rest of the entities accessed by the transaction. It is not important to which subject database they belong. This means that, in practice, the value of H will be at most 2 or 3. The variable H was defined in the previous chapter as the number of edges in the transaction tree. Here H is the number of edges that connect closely related entities of a transaction.

5.3. High-Level Logical Design

During high-level logical design, we shall partition a given data model into subject databases. The inputs to this step consist of:

1. an Entity-Relationship diagram,
2. entity weights,
3. a set of transactions with their associated complexity costs, and
4. limit on the size of subject databases.

Previously, we noted that the problem of optimal partitioning of a data model into subject databases is NP-hard. We also analyzed the properties of absolute approximation and ϵ-approximation heuristics for solving partitioning problems and the properties of several partitioning and clustering techniques. The conclusion of this analysis is not very encouraging. So far, there is no good proven technique to solve the partitioning problem. This means that, for years to come, database designers will have to rely mainly on their intuition in order to make certain decisions. Still, it does not mean that science can offer nothing to help database designers. One way to go is to split the problem into several steps, some of which can be solved automatically by using polynomial algorithms. To implement this approach, we can use the developed algorithm for partitioning trees. This algorithm will work if an Entity-Relationship diagram is represented by a graph without cycles and each transaction is also represented by a graph without cycles. In practice, graphs representing ER digrams usually contain relatively few cycles, and transaction graphs usually have no cycles at all. This means that the problem of partitioning the ER diagram is reasonably close, in some intuitive sense, to the problem of tree partitioning. Intuitively, we can expect that if we break a few cycles in the ER diagram and then apply the tree partitioning algorithm, the resulting partition will be reasonably good.

The partitioning algorithm will perform reasonably if the values of parameters D and H are very small. In practical cases, we can certainly expect this. We have already noted that the value of H is unlikely to be greater than 3. On average, the value of D is also unlikely to exceed 3. This means that we have very good reasons to believe that the application of the tree partitioning algorithm will be successful for the data model partitioning problem.

The removal of cycles will be based mainly on the common sense of a database designer. First, the database designer should analyze each transaction and destroy all detected cycles by removing the least important edges. At the same time, the database designer should strip transaction trees from loosely related entities. This new set of transaction trees will be used as an input to the partitioning algorithm.

The removal of cycles from an ER diagram can be done in the same way, although we can suggest some procedure for doing it. First, we should note that for the purpose of partitioning the ER diagram, the direction of edges or presence of multiple edges in the graph representing the ER diagram are not important. So, for the purpose of partitioning the ER diagram, we shall consider the undirected graph $G_u(V_u,E_u)$ without multiple edges instead of the graph $G(V,E)$ representing the ER diagram. The undirected graph $G_u(V_u,E_u)$ can be constructed from the graph $G(V,E)$ by replacing directed edges with undirected ones and replacing multiple edges with a single one. We shall also assign weights for all edges of the undirected graph $G_u(V_u,E_u)$ such that the weight of each edge will be equal to the total cost of all transactions navigating through this edge. After this, we shall find the maximal weight spanning tree of the graph $G_u(V_u,E_u)$. The spanning tree obtained will not contain certain edges which are contained in the graph $G_u(V_u,E_u)$. The edges removed, however, can be part of some transaction trees. These transaction trees will be deleted from the input to the partitioning algorithm. We have decided to use the maximal

weight spanning tree of the ER diagram and hope that it will soften the impact of deleting some transactions from the consideration. This spanning tree and the rest of the transaction trees will constitute an input to the tree partitioning algorithm.

5.4. Physical Database Design

The high-level logical design described in the previous section will provide us with subject databases. The physical database design process will be applied independently to each of the resulting subject databases. The main purpose of physical database design is to identify physical clustering of the records representing the entities in order to minimize the number of input/output operations. In order to carry out physical database design, we should first reestablish all edges in the graphs representing subject databases in the same way as they appear in the ER diagram. In other words, we shall consider ER diagrams for all subject databases. We shall also consider the original set of transactions, where each transaction is assigned a weight which is equal to the frequency of this transaction. After that, we shall assign a weight to each edge of the ER diagrams which is equal to the total weight of all transactions navigating through this edge. All this will constitute the input to the physical database design. In practice, the physical database design can be conducted in the following way:

1. Replace all multiple edges between two entities by an edge having the highest weight.
2. Examine all one-to-one relationships between entities, that is, all undirected edges of the ER diagrams. If the weight of an undirected edge between two entities is high, then merge these two entities into one, otherwise delete the edge. As a result of this we shall obtain a pure directed graph, where each edge represents a one-to-many relationship.
3. Identify all entities which have two or more incoming edges. Remove all these edges except the one having the highest weight.

4. Destroy all cycles by deleting the edges having the lowest weight. As a result, we obtain a set of directed trees, where a parent node is connected to its child by an edge directed from a parent node to a child node.

5. Each of the trees obtained in Step 4 can be partitioned further into smaller trees, each representing a separate file. The purpose of this partitioning is to minimize the number of I/O accesses in the resulting database. The algorithm for this type of partitioning was developed by M. Schkolnick [20] in 1977.

The records in the files determined in Step 5 should be placed in a hierarchical order according to their associated trees.

Chapter 6

Conclusions

Partitioning a set of objects into several subsets of closely related objects is at the heart of many applications. We have shown that in order to reduce system maintenance cost it is necessary to partition a set of entities into several subject databases with a minimal interface.

We have considered the general partitioning problem with no restriction on the structure of transactions and with no restriction on the properties of an entity set. We have found that the general partitioning decision problem is NP-complete; the problem of finding an absolute approximation to the general partitioning optimization problem is NP-hard; and the problem of finding an ϵ-approximation to the general partitioning optimization problem with a limited number of subject databases, with $\epsilon < 1$ and with weighted entities is also NP-hard. The problem of finding an ϵ-approximation to the general partitioning optimization problem with an unlimited number of subject databases, or with $\epsilon > 1$, or with weights of all entities equal to one remains open. It is still not clear whether this problem is NP-hard or not.

Also, we have discussed the bipartite partitioning problem, where a set of entities consists of two disjoint subsets and each transaction accesses only two entities, each from a different subset. We have found that these restrictions do not make the problem any "easier".

Finally, we have analyzed the problem of partitioning a set of entities into subject databases where each entity is a node of a data model tree, each transaction is represented by a connected subgraph of the data model tree, and each subject database is a connected subgraph of the data model tree. We have found a polynomial time algorithm to solve this problem and have analyzed the time complexity of this algorithm. We have reviewed all stages of database design and have shown how our

algorithm can be integrated into the design process.

The present work left open several other interesting issues. One of them is how to select a proper bound on the size of subject databases. Another interesting problem is related to the tree partitioning algorithm. Suppose, we have a set of entities and a set of transactions each represented by a subset of entities. The problem is to determine whether it is possible or not to represent the set of entities as nodes of some tree, such that all transaction subsets could be represented as connected subgraphs of this tree.

References

[1] Vladimir Arlazarov and Misha Donskoy, Issues in Information Systems, *Talk at the University of Alberta*, October 1986 .

[2] E. F. Codd, "A Relational Model of Data for Large Data Banks", *Comm. ACM*, Vol. 13, No. 6, June 1970, pp. 377-387.

[3] E. F. Codd, Further Normalization of the Data Base Relational Model, *in R. Rustin ed. Data Base Systems*, 1972, pp. 39-64.

[4] S. Cook, The Complexity of Theorem Proving Procedures, *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 1971, pp. 151-158.

[5] B. S. Duran and P. L. Odell, *Cluster Analysis*, Springer Verlag, New York, 1971.

[6] M. Garey and D. Johnson, Computers and Intractability, 1979.

[7] T. Gonzalez and S. Sahni, P-complete approximation problems, *J. ACM*, 1976, pp. 555-565.

[8] L. L. Gorinshteyn, "The Partitioning of Graphs", *Engineering Cybernetics*, Vol. 7, No. 1, January 1969, pp. 76-82.

[9] E. Horowitz and S. Sahni, Fundamentals of Computer Algorithms, 1978.

[10] S. C. Johnson, "Hierarchical Clustering Schemes", *Psychometrika*, Vol. 32, 1967, pp. 241-254.

[11] J. Kaoprzyk and W. Stanozak, "On an Extension of the Method of Minimally Interconnected Subnetworks", *Control and Cybernetics*, Vol. 4, 1976, pp. 61-77.

[12] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *The Bell System Technical Journal*, Vol. 49, No. 2, February 1970, pp. 291-307.

[13] B. W. Kernighan, "Optimal Sequential Partitions of Graphs", *J. Assoc. Comput. Mach.*, Vol. 18, No. 1, January 1971, pp. 34-40.

[14] S. Lin, "Computer Solutions of the Travelling Salesman Problem", *B.S.T.J.*, Vol. 44, No. 10, December 1965, pp. 2245-2269.

[15] F. Luccio and M. Sami, "On the Decomposition of Networks in Minimally Interconnected Subnetworks", *IEEE Transactions on Circuit Theory*, Vol. 16, No. 2, 1969, pp. 184-188.

[16] J. A. Lukes, "Efficient Algorithm for Partitioning of Trees", *IBM Journal of Research and Development*, Vol. 19, May 1974, pp. 217-224.

[17] J. A. Lukes, "Combinatorial Solution to the Partitioning of General Graphs", *IBM Journal of Research and Development*, Vol. 19, No. 2, March 1975, pp. 170-180.

[18] J. Martin, *Strategic Data-planning Methodology*, Prentice Hall, N.J., 1982.

[19] W. T. McCormick, P. J. Sweitzer and T. W. White, "Problem Decomposition and Data Reorganization by a Clustering Technique", *Operations Research*, Vol. 20, No. 5, September 1972, pp. 993-1009.

[20] M. Schkolnick, "A Clustering Algorithm for Hierarchical Structures", *ACM Transactions. Database Systems.*, Vol. 2, No. 1, 1977, pp. 27-44.

[21] J. D. Ullman, *Principles of Database Systems*, Computer Science Press, 1980.

[22] E. Yourdon, *Managing the Structured Techniques*, Yourdon Press, New York, 1979.

[23] J. Zupan, *Clustering of Large Data Sets*, Research Studies Press, New York, 1972.

Appendix A1

Solution of the Differential Equation

In this section, we shall find a time independent solution of the following differential equation:

$$\dot{W} = -kW + f\frac{S-W}{S}P$$

where k and f are positive coefficients and performance P is

$$P = \frac{S}{Wt + T(S-W)}.$$

First, let us set $R = \frac{W}{S}$ and rewrite the differential equation and expression for P as follows:

$$\dot{SR} = -kRS + f\frac{1-R}{Rt + T(1-R)}$$

$$P = \frac{1}{Rt + (1-R)T}.$$

The points of equilibrium of this differential equation can be found from the equation

$$-kRS + f\frac{1-R}{tR + T(1-R)} = 0.$$

Rewriting the last equation in the canonical form, we obtain

$$kS(T-t)R^2 - (kST + f)R + f = 0.$$

This quadratic equation has two roots:

$$R_1 = \frac{kST + f + (k^2S^2T^2 - 2kSTf + f^2 + 4kStf)^{\frac{1}{2}}}{2kS(T-t)}$$

$$R_2 = \frac{kST + f - (k^2S^2T^2 - 2kSTf + f^2 + 4kStf)^{\frac{1}{2}}}{2kS(T-t)}$$

Let us look at the behaviour of the roots R_1 and R_2 for large values of S. Expanding the expressions for R_1 and R_2 in a Taylor series using $\frac{1}{S}$ as a small parameter, we obtain

$$R_1 = \frac{kST + \frac{t}{T}f}{kS(T-t)} > 1$$

$$R_2 = \frac{f}{kST}.$$

The root R_1 should be discarded because $R_1 > 1$. Now, using R_2 as an equilibrium point of the differential equation, we can see that

$$R = \frac{W}{S} = \frac{f}{kST}$$

and

$$W = \frac{f}{kT}.$$

Finally, we can calculate the performance P of the maintenance analyst as

$$P = \frac{1}{\frac{ft}{kST} + (1 - \frac{f}{kST})T}.$$

We shall expand the expression for P in a Taylor series using $\frac{1}{S}$ as a small parameter. This will give us the following expression for the performance P:

$$P = \frac{1}{T}(1 + \frac{f(T-t)}{kST^2}).$$

Appendix A2

Example of Partitioning of the Tree.

This appendix contains an example of partitioning a data model tree without a limit on the number of subject databases. The example of the data model tree is given in Figure 4.1, and the transaction trees are given in Figure 4.3. We shall use the same notations as defined in Chapter 4. Also, we shall use set a as the intersection of a root cluster of the partition with the corresponding influence set, and V as the value of the partition. We assume that all nodes have weights equal to one, and the maximal weight of the partition cluster is 3.

We start with trivial partitions of the trees $R(1,0)$, $R(2,0)$, $R(3,0)$, and $R(4,0)$.

PARTITION $R(1,0)$

$I(1,0) = (1)$

$a_1(1,1,0) = (1)$

$P(a_1(1,1,0)) = (1); \quad V=0$

$P(1,0) = (1); \quad V=0$

PARTITION $R(2,0)$

$I(2,0) = (2)$

$a_1(1,2,0) = (2)$

$P(a_1(1,2,0)) = (2); \quad V=0$

$P(2,0) = (2); \quad V=0$

PARTITION $R(3,0)$

$I(3,0) = (3)$

$a_1(1,3,0) = (3)$

$P(a_1(1,3,0)) = (3); \quad V=0$

$P(3,0) = (3); \quad V=0$

PARTITION $R(4,0)$

$I(4,0) = (4)$

$a_1(1,4,0) = (4)$

$P(a_1(1,4,0)) = (4); \quad V=0$

$P(4,0) = (4); \quad V=0$

PARTITION $R(5,0)$

$I(5,0) = (5)$

$a_1(1,5,0) = (5)$

$P(a_1(1,5,0)) = (5); \quad V=0$

PARTITION $R(5,1)$

$I(5,1) = (5,2)$

1. Weight of the root cluster is 1.

    1.1. CONCATENATE($P(a_1(1,5,0))$, $P(2,0)$) = (5)(2); $V=0$

        $a = (5)$

        $a_1(1,5,1) = (5)$

        $P(a_1(1,5,1)) = (5)(2); \quad V=0$

2. Weight of the root cluster is 2.

    2.1. MERGE ($P(a_1(1,5,0))$, $P(a_1(1,2,0))$) = (5,2); $V=0$

        $a = (5,2)$

        $a_1(2,5,1) = (5,2)$

        $P(a_1(2,5,1)) = (5,2); \quad V=0$

PARTITION $R(5,2)$

$I(5,2) = (5,2)$

1. Weight of the root cluster is 1.

   1.1 CONCATENATE($P(a_1(1,5,1))$, $P(3,0)$) = (5)(2)(3); $V=0$

   a = (5)

   $a_1(1,5,2)$ = (5)

   $P(a_1(1,5,2))$ = (5)(2)(3); $V=0$

2. Weight of the root cluster is 2.

   2.1. CONCATENATE($P(a_1(2,5,1))$, $P(3,0)$) = (5,2)(3); $V=0$

   a = (5,2)

   $a_1(2,5,2)$ = (5,2)

   $P(a_1(2,5,2))$ = (5,2)(3); $V=0$

   2.2. MERGE($P(a_1(1,5,1))$, $P(a_1(1,3,0))$) = (5,3)(2); $V=1$

   a = (5)

   $a_2(2,5,2)$ = (5)

   $P(a_2(2,5,2))$ = (5,3)(2); $V=1$

3. Weight of the root cluster is 3.

   3.1. MERGE($P(a_1(2,5,1))$, $P(a_1(1,3,0))$) = (5,3,2); $V=1$

   a = (5,2)

   $a_1(3,5,2)$ = (5,2)

   $P(a_1(3,5,2))$ = (5,3,2); $V=1$

PARTITION $R(5,3)$

$I(5,3)$ = (5,4)

1. Weight of the root cluster is 1.

   1.1. CONCATENATE($P(a_1(1,5,2))$, $P(4,0)$) = (5)(2)(3)(4); $V=0$

   a = (5)

   $a_1(1,5,3)$ = (5)

$P(a_1(1,5,3)) = (5)(2)(3)(4)$; $V=0$

2. Weight of the root cluster is 2.

2.1. CONCATENATE($P(a_1(2,5,2))$, $P(4,0)$) = $(5,2)(3)(4)$; $V=0$

$a = (5)$

$a_1(2,5,3) = (5)$

$P(a_1(2,5,3)) = (5,2)(3)(4)$; $V=0$

2.2. CONCATENATE($P(a_2(2,5,2))$, $P(4,0)$) = $(5,3)(2)(4)$; $V=1$

$a = (5)$

OPTIMALITY-CHECK shows that this partition should replace in the set $Q(5,3)$ partition $P(a_1(2,5,3))$ obtained in Step 2.1. For this new partition we have:

$a_1(2,5,3) = (5)$

$P(a_1(2,5,3)) = (5,3)(2)(4)$; $V=1$

2.3. MERGE ($P(a_1(1,5,2))$, $P(a_1(1,4,0))$) = $(5,4)(2)(3)$; $V=0$

$a = (5,4)$

$a_2(2,5,3) = (5,4)$

$P(a_2(2,5,3)) = (5,4)(2)(3)$; $V=0$

3. Weight of the root cluster is 3.

3.1. CONCATENATE ($P(a_1(3,5,2))$, $P(4,0)$) = $(5,3,2)(4)$; $V=1$

$a = (5)$

$a_1(3,5,3) = (5)$

$P(a_1(3,5,3)) = (5,3,2)(4)$; $V=1$

3.2. MERGE ($P(a_1(2,5,2))$, $P(a_1(1,4,0))$) = $(5,2,4)(3)$; $V=2$

$a = (5,4)$

$a_2(3,5,3) = (5,4)$

$P(a_2(3,5,3)) = (5,2,4)(3); \quad V=2$

3.3. MERGE ($P(a_2(2,5,2))$, $P(a_1(1,4,0))$) = (5,3,4)(2); $V=1$

$a = (5,4)$

$a_2(3,5,3) = (5,4)$

OPTIMALITY-CHECK will reject this partition because its value is lower than the value of the partition $P(a_2(3,5,3))$ obtained in Step 3.2.

Optimal partition $P(5,3)$ is (5,2,4)(3); $V = 2$

For the rest of the trees $R(v,i)$ we shall list only partitions belonging to the sets $Q(v,i)$ without intermediary results.

PARTITION $R(6,0)$

$I(6) = (6)$

$P = (6)$; $a = (6)$; $V = 0$

PARTITION $R(6,1)$

$I(6,1) = (6,1)$

1. Weight of the root cluster is 1.

   1.1. $P = (6)(1)$; $a = (6)$; $V = 0$

2. Weight of the root cluster is 2.

   2.1. $P = (6,1)$; $a = (6,1)$; $V = 0$

PARTITION $R(6,2)$

$I(6,2) = (6,1,5)$

1. Weight of the root cluster is 1.

1.1. $P = (6)(1)(5,2,4)(3)$; $a = (6)$; $V = 0$

2. Weight of the root cluster is 2.

2.1. $P = (6,5)(1)(2)(4)(3)$; $a = (6,5)$; $V = 0$

2.2. $P = (6,1)(2,5,4)(3)$; $a = (6,1)$; $V = 2$

3. Weight of the root cluster is 3.

3.1. $P = (6,5,4)(1)(2)(3)$; $a = (6,5)$; $V = 3$

3.2. $P = (6,5,1)(2)(3)(4)$; $a = (6,5,1)$; $V = 1$

Optimal partition of the tree $R(6,2)$ is $(6,5,4)(1)(2)(3)$ with the value 3.

PARTITION $R(7,0)$

$I(7,0) = (7)$

$P = (7)$; $a = (7)$; $V = 0$

PARTITION $R(8,0)$

$I(8,0) = (8)$

$P = (8)$; $a = (8)$; $V = 0$

PARTITION $R(8,1)$

$I(8,1) = (8,6)$

1. Weight of the root cluster is 1.

1.1. $P = (8)(6,5,4)(1)(2)(3)$; $a = (8)$; $V = 3$

2. Weight of the root cluster is 2.

2.1. $P = (8,6)(5,2,4)(1)(3)$; $a = (8,6)$; $V = 2$

3. Weight of the root cluster is 3.

3.1. $P = (8,6,1)(5,2,4)(3)$; $a = (8,6)$; $V = 4$

PARTITION $R(8,2)$

Finally, the optimal partition of the tree is

$(8,7,6)(5,2,4)(1)(3)$ with the value $V = 4$.